МИНИСТЕРСТВО НАУКИ, ВЫСШЕГО ОБРАЗОВАНИЯ И ИННОВАЦИЙ КЫРГЫЗСКОЙ РЕСПУБЛИКИ
КЫРГЫЗСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ им.И.АРАБАЕВА
ОСПО ИНСТИТУТА НОВЫХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

«УТВЕРЖДАЮ»
Директор ИНИТ
КГУ им.И.Арабаева
к.т.н., и.о.доц. У.Керимов
«04» 09 2025г.

УЧЕБНО-МЕТОДИЧЕСКИЙ КОМПЛЕКС

по дисциплине: "Алгоритмизация и программирование"
для студентов специальности: 230701 – «Прикладная информатика (по отраслям)
форма обучения: очное

Учебно-методический комплекс составлен на основе Государственного Образовательного Стандарта среднего профессионального образования КР

Учебно-методический комплекс разработала: преподаватель отделения СПО ИНИТ КГУ имени И. Арабаева Тыналиева Чынара Таласбековна

Бишкек 2025 г.

•Введение
•Цели и задачи дисциплины, ее значение в учебном процессе
•Межпредметные связи. Перечень дисциплин и их разделов, усвоение которых необходимо при изучении данной дисциплины
•Компетенции по Госстандарту
•Структура дисциплины
•Методическая разработка лекций по дисциплине (Краткий курс лекций, презентации)
•Методическая разработка аудиторных форм работы (Краткое содержание практических занятий)
•Учебно-методические материалы
•Критерии баллов — рейтинговой оценки знаний и умений студентов
•Формы текущего и итогового контроля
•Вопросы к модулям
•Темы для самостоятельной работы студентов
•Тестовые задания
•Список литературы и учебно-методическая литература по дисциплине, разработанная преподавателями отделения
•Глоссарий

МИНИСТЕРСТВО НАУКИ, ВЫСШЕГО ОБРАЗОВАНИЯ И ИННОВАЦИЙ КЫРГЫЗСКОЙ РЕСПУБЛИКИ
КЫРГЫЗСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ им. И.АРАБАЕВА
ОСПО ИНСТИТУТА НОВЫХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

«УТВЕРЖДАЮ»
Директор ИНИТ
КГУ им. И. Арабаева
к.т.н., и. о. доц. У.Т. Керимов
_____
«01» 09 2025г.

РАБОЧАЯ ПРОГРАММА

по дисциплине: **«Алгоритмизация и программирование»**
для студентов специальности: 230701 – «Прикладная информатика (по отраслям)
форма обучения: очное
институт: ИНИТ
отделение: ОСПО ИНИТ
курс: 1
семестр: 2
экзамен (семестр): 2
всего часов по учебному плану: 82
из них:

-лекции:44
-практикалык: 28
-прием и контроль:10
-самостоятельная работа: 48

Рабочая программа составлена в соответствии с требованиями Государственного Образовательного Стандарта среднего профессионального образования КР
Рабочую программу разработала: преподаватель отделения СПО ИНИТ КГУ имени И. Арабаева Тыналиева Чынара Таласбековна

Рассмотрена и утверждена на заседании ОСПО ИНИТ КГУ им. И. Арабаева Протокол № 1 от «02» 09 2024г. Зав. отделением: Н.С. Сейтказиева	Одобрено учебно-методическим советом ИНИТ КГУ им. И. Арабаева Протокол № 1 от «04» 09 2024г. Председатель УМС ИНИТ:

Введение

Рабочая программа учебной дисциплины «Алгоритмизация и программирование» предназначена для предоставление студентам важные навыки и знания, которые могут быть полезными в различных областях, и служит введением в мир программирования и информационных технологий. Рабочая программа составлена на основании государственного стандарта по данной специальности.

Освоение основ программирования: для студентов, которые только начинают изучать программирование, эта дисциплина предоставляет введение в основы программирования на языке Python. Python выбирается, потому что он имеет простой синтаксис, что делает его доступным для начинающих, и в то же время он является мощным инструментом для разработки различных типов приложений.

Освоение алгоритмических принципов: студенты учатся разрабатывать алгоритмы для решения разнообразных задач. Это включает в себя понимание понятий, таких как последовательность, ветвление и циклы, а также работу с данными и структурами данных.

Практические навыки: Студенты изучают, как применять полученные знания в практике. Они пишут и отлаживают программы на Python, решая реальные задачи, что помогает им развивать навыки программирования и решения задач.

Подготовка к более сложным курсам: Дисциплина может служить основой для более продвинутых курсов по программированию и разработке, предоставляя студентам необходимые навыки и знания.

Развитие логического мышления: Работа с алгоритмами и программирование способствуют развитию логического мышления и способности анализа задач, что полезно в различных аспектах жизни и карьеры.

Таким образом, дисциплина "Алгоритмизация и программирование " предоставляет студентам важные навыки и знания, которые могут быть полезными в различных областях, и служит введением в мир программирования и информационных технологий.

Цели и задачи дисциплины, ее значение в учебном процессе

Цель изучения дисциплины

Целью дисциплины «Алгоритмизация и программирование» является: развития у студентов навыков и понимания, необходимых для создания компьютерных программ с использованием алгоритмов и программных языков. Основной упор делается на развитие следующих аспектов:

1)Разработка алгоритмического мышления: студенты учатся разбираться в сложных задачах и разбивать их на более простые подзадачи. Они также изучают, как разрабатывать эффективные алгоритмы для решения различных задач.

2) Освоение основ программирования: целью дисциплины является овладение основами программирования на выбранном языке, выбранный язык программирования Python. Это включает в себя понимание синтаксиса, структур данных, операторов и функций языка программирования.

3) Практические навыки: студенты изучают, как писать программы, решающие реальные задачи. Они пишут код, отлаживают его и тестируют, что развивает практические навыки программирования.

4) Подготовка к более продвинутым курсам: дисциплина может служить базой для более продвинутых курсов по программированию и разработке, помогая студентам строить на этом фундаменте более сложные навыки и знания.

5)Развитие креативности: разработка программ требует креативности и способности находить инновационные решения. Целью дисциплины также может быть развитие этой креативности и умения находить нестандартные способы решения задач.

Итак, цель изучения дисциплины "Алгоритмизация и программирование" заключается в подготовке студентов к созданию программного обеспечения, развитии логического и алгоритмического мышления, а также приобретении практических навыков, которые могут быть полезными в различных сферах деятельности.

Задачи изучения дисциплины

Изучение дисциплины "Алгоритмизация и программирование" включает в себя ряд задач, которые студенты должны выполнять для достижения основных целей и компетенций. Вот некоторые из типичных задач, которые могут быть поставлены перед студентами в рамках этой дисциплины:

освоение синтаксиса Python: студенты должны овладеть основами синтаксиса Python, включая правила написания переменных, операторов, условных выражений и циклов.

разработка простых программ: создание простых программ для решения конкретных задач. Это может включать в себя программы для вычисления математических формул, преобразования единиц измерения, обработки строк и списков.

алгоритмические задачи: решение задач, которые требуют разработки алгоритмов, таких как сортировка, поиск, анализ данных и др. Студенты должны научиться разрабатывать эффективные алгоритмы для решения различных задач.

работа с функциями: понимание и использование функций в Python. Создание собственных функций и их вызов в программе.

работа с данными: операции с различными типами данных, включая числа, строки, списки и словари. Извлечение данных из файлов и их обработка.

управление ошибками и отладка: навыки обнаружения и устранения ошибок в программах. Использование инструментов отладки.

разработка игр и графических приложений: введение в создание простых игр или графических приложений, чтобы продемонстрировать применение программирования в развлекательных и креативных целях.

работа с библиотеками и фреймворками: знакомство со сторонними библиотеками и фреймворками Python для более сложных задач, таких как веб-разработка, машинное обучение или анализ данных.

командная работа: участие в проектах, которые требуют совместной разработки программного обеспечения, что помогает студентам развивать навыки командной работы и управления проектами.

создание портфолио: важной задачей также может быть создание портфолио, в котором студенты демонстрируют свои проекты и достижения в программировании на Python.

Эти задачи позволяют студентам развивать свои навыки программирования, логического мышления и понимания алгоритмов, что является основой для дальнейшей работы в области информационных технологий и программирования.

Межпредметные связи. Перечень дисциплин и их разделов, усвоение которых необходимо при изучении данной дисциплины.

Изучение дисциплины "Алгоритмизация и программирование" включает в себя множество межпредметных связей, так как это ключевой предмет в информатике и компьютерных науках. Вот перечень дисциплин и их разделов, которые имеют межпредметные связи с алгоритмизацией и программированием:

1.Математика:

- Дискретная математика: Изучение математических структур, как графов и теории чисел, является важным для разработки эффективных алгоритмов.
- Линейная алгебра: Матрицы и векторы используются в компьютерной графике и обработке данных.

2.Основы информатики: "Алгоритмизация и программирование" часто вводит студентов в основы информатики, такие как структуры данных, алгоритмы и основные принципы обработки информации.

3.Операционные системы:

Программирование под операционными системами, работа с процессами, потоками и системными вызовами.

4.Компьютерная архитектура:

Основы архитектуры компьютеров: Понимание работы процессоров, памяти и устройств ввода-вывода.

4.Базы данных:

Знание баз данных и языка SQL, как учебной части "Алгоритмизации и программирования", важно для создания и управления базами данных.

5.Операционные системы:

Программирование под операционными системами: Взаимодействие с операционной системой, управление процессами и потоками.

Это лишь небольшой перечень предметов и разделов, связанных с "Алгоритмизацией и программированием". Понимание этих предметов и их межпредметные связи помогает студентам более полно освоить алгоритмизацию и программирование, так как они представляют собой важные основы информатики и компьютерных наук.

Компетенции по Госстандарту

В соответствии с целями ОПОП и задачами профессиональной деятельности, должен обладать следующими компетенциями:

а) универсальными:

общенаучными(ОК):

- уметь организовывать собственную деятельность, выбирать методы и способы выполнения профессиональных задач, оценивать их эффективность и качество(ОК-1);
- решать проблемы, принимать решения в стандартных и нестандартных ситуациях, проявлять инициативу и ответственность(ОК-2);
- осуществлять поиск, интерпретацию и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития(ОК-3);
- способен приобретать новые знания, с большой степенью самостоятельности, с использованием современных образовательных и информационных технологий(ОК-9);
- способен на научной основе оценить свой труд, оценивать с большой степенью самостоятельности, результаты своей деятельности(ОК-10);

профессиональными(ПК):

- обрабатывать статический информационный контент(ПК-1);
- обрабатывать динамический информационный контент(ПК-2);
- осуществлять подготовку оборудования к работе(ПК-3);
- контролировать работу компьютерных, периферийных устройств и телекоммуникационных систем, обеспечивать их правильную эксплуатацию(ПК-5);

Объем дисциплины и виды учебной работы

Виды учебной работы	Всего часов
Общая трудоемкость дисциплины	130
Аудиторные занятия	82
Лекции	44
Практические занятия	28
Прием и контроль	10
Самостоятельная работа	48
Вид итогового контроля (зачет, экзамен)	экзамен

Структура дисциплины

План изучения дисциплины (лекциии I и II семестр):

№	Темы дисциплины	Кол.часов
	Глава №1.Теоретические основы алгоритмизации и программирования	
1.	Алгоритм.Свойства алгоритма. Формы записи алгоритмов. Назначение функциональных блоков.	**2**
2.	Основные этапы решения задач. Основные структуры алгоритмов. Данные и их типы.	**2**
3.	Языки программирования. Классификация языков программирования. Методы и принципы программирования. Виды программного обеспечения (ПО).	**2**
	Глава №2. Язык программирования PYTHON	

4.	Введение в язык программирования Python. Среда Python. Первый запуск рабочей среды. Основные элементы языка Python.	**2**
5.	Операции, переменные, литералы, типы данные в Python. Ввод и вывод данных в программах на языке Python. Операторы и выражения.	**2**
	Глава № 3. Основные алгоритмические инструкции языка Python.	
6.	Линейные алгоритмы: операции над целочисленными, вещественными, логическими данными.	**2**
7.	Готовые модули Math, Random в Python.	**2**
8.	Разветвляющийся алгоритм: Простой условный оператор, Сокращенный условный оператор, Составной условный оператор.	**2**
9.	Многозначные ветвления. Алгоритмы поиска максимального и минимального элементов.	**2**
10.	Циклический алгоритм: Оператор цикла While. Инструкции break и continue.	**2**
11.	Оператор цикла For. Функция range().	**2**
12.	Вложенные циклы.	**2**
	Глава № 4. Коллекции в Python.	
13.	Работа со строками: основные понятия, операции с ними. Методы строк.Форматирование строк. Базовые алгоритмы обработки строк.	**2**
14.	Работа с множествами в Python.	**2**
15.	Работа со списками в Python.	**2**
16.	Работа с кортежами в Python.	**2**
17.	Работа со словарями в Python.	**2**
18.	Обработка вложенных последовательностей (ВП). Базовые алгоритмы ВП.	**2**
	Глава №5. Работа с функциями. Создание модулей.	

19.	Работа с функциями: создание пользовательских функций.	**2**
20.	Создание модулей.	**2**
	Глава №6. Работа с графикой в Python.	
21.	Рисования с помощью модуля Turtle.	**2**
22.	Использование функций в модуле Turtle.	**2**
	Всего:	**44**

План изучения дисциплины(практические):

№	Темы практических работ	Кол.часов
	Практическая работа №1	
1.	Введение в язык программирования Python.Изучить основные типы данных, команды ввода и вывода данных.	**2**
	Практическая работа №2	
2.	Линейный алгоритм: Математические операции в Python.Библиотека (модуль) math	**2**
	Практическая работа №3	
3.	Разветвляющиеся алгоритм: Структура ветвление в Python.Множественное ветвление	**2**
	Практическая работа №4	
4.	Циклический алгоритм: Работа с циклами в Python:Цикл while в Python.	**2**
	Практическая работа №5	
5.	Цикл for в Python.Работа с вложенными циклами.	
	Практическая работа №6	
6.	Работа со строками в Python. Методы работы со строками	**2**

	Практическая работа №7	
7.	Работа с множествами	**2**
	Практическая работа №8	
8.	Работа со списками. Операции над списками в Python	**2**
	Практическая работа №9	
9.	Работа с кортежами. Классические способы обработки кортежей.	**2**
	Практическая работа №10	
10.	Работа со словарями	**2**
	Практическая работа №11	
11.	Работа над формированием вложенных последовательностей	**2**
	Практическая работа №12	
12.	Работа над создание пользовательских функций	**2**
	Практическая работа №13	
13.	Работа над созданием модулей	**2**
	Практическая работа №14	
14.	Работа с графикой: Рисование с помощью модуля Turtle	**2**
	Всего:	**28**

Методическая разработка лекций по дисциплине (Краткий курс лекций, презентации)

Глава 1.Теоретические основы алгоритмизации и программировании

В этой главе рассматриваются базовые понятия, относящиеся к любому процедурному языку программирования высокого уровня. Основное внимание уделяется структурной методике построения алгоритмов: использованию базовых алгоритмических структур, выделению в решаемой задаче подзадач и составлению вспомогательных алгоритмов. На этой основе можно переходить к изучению любого процедурного языка, поддерживающего структурную методику.

Тема 1.1. Алгоритм и его свойства

1. Понятие алгоритма и его свойства

Само слово "алгоритм" происходит от имени персидского математика Аль Хорезми, который в IX веке разработал правила четырех арифметических действий (сегодня мы бы сказали алгоритмы арифметических действий).

В начале XX века алгоритмы стали объектом изучения математиков, появились различные математические уточнения понятия "алгоритм" и возникла целая отрасль математики – теория алгоритмов. Результаты, полученные теорией алгоритмов, служат теоретическим фундаментом всей компьютерной технологии, но в повседневной программистской практике не используются.

Алгоритм – это описание некоторой последовательности действий, приводящее к решению поставленной задачи.

Алгоритм – система четких однозначных указаний, которая определяет последовательность действий над некоторыми объектами и после конечного числа шагов приводит к получению требуемого результата.

Алгоритмы бывают численными и логическими.

Алгоритмы, в соответствии с которыми решение поставленных задач сводится к арифметическим действиям, называются **численными алгоритмами.**

Алгоритмы, в соответствии с которыми решение поставленных задач сводится к логическим действиям, называются *логическими алгоритмами* (алгоритмы поиска минимального числа, поиска пути в лабиринте).

Основными свойствами алгоритма являются:

1) Дискретность – разделение выполнения решения задачи на отдельные операции.

Под **дискретностью** понимается то, что алгоритм состоит из описания последовательности шагов обработки, организованных таким образом, что в начальный момент задается исходная ситуация, а после каждого следующего шага ситуация преобразуется на основе данных, полученные в предшествующие шаги обработки. Дискретность алгоритма означает, что он исполняется по шагам: каждое действие, предусмотренное алгоритмом,

исполняется только после того, как закончилось исполнение предыдущего, то есть преобразование исходных данных в результат происходит во времени дискретно.

2) Детерминированность (определенность) – каждая команда алгоритма должна однозначно определять действия исполнителя.

Это свойство означает, что на каждом шаге алгоритма однозначно определяется преобразование данных, полученных на предшествующих шагах алгоритма, то есть на одинаковых исходных данных алгоритм должен всегда давать одинаковые результаты.

3) Результативность (конечность) - завершение работы алгоритма за конечное число шагов (при этом количество шагов может быть заранее неизвестным и различным для разных исходных данных).

4) Массовость (универсальность) - алгоритм решения задачи разрабатывается в общем виде, то есть возможность решения класса задач, различающихся лишь исходными данными. При этом исходные данные выбираются из некоторой области, называемой областью применимости алгоритма.

5) Понятность – содержание допустимого набора команд, понятного конкретному исполнителю. Каждый шаг алгоритма должен обязательно представлять собой какое-либо допустимое действие, т.е. алгоритм строится для конкретного исполнителя автором и должен быть им обоим понятен. Это облегчает проверку и модификацию алгоритма при необходимости.

Тема 1.2. Формы записи алгоритмов

Процесс составления алгоритмов называют **алгоритмизацией**.

Алгоритм, реализующий решение задачи, можно представить различными способами – с помощью графического или текстового описания.

Графический способ представления алгоритмов имеет ряд преимуществ благодаря визуальности и явному отображению процесса решения задачи. Алгоритмы, представленные графическими средствами, получили название **блок-схем**.

Текстовое описание алгоритма является достаточно компактным и может быть реализовано на **естественном языке** или **специальном (алгоритмическом) языке** в виде программы.

Все три способа представления алгоритмов можно считать взаимодополняющими друг друга. На этапе проектирования алгоритмов наилучшим способом является графическое представление, а на этапах проверки и применения алгоритма – текстовая запись в виде программы.

Правила выполнения блок-схем:

Блок-схемой называется наглядное изображение алгоритма, когда отдельные действия (этапы алгоритма) изображаются при помощи различных геометрических фигур (блоков), а связи между этапами (последовательность выполнения этапов) указываются при помощи стрелок, соединяющие эти фигуры.

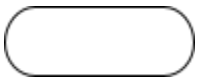
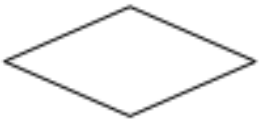
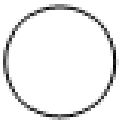
Выполнение блок-схем осуществляется по ГОСТ 19.701–90.

При выполнении блок-схем внутри каждого блока указывается поясняющая информация, которая характеризует действия, выполняемые этим блоком. Потоки данных в схемах показываются линиями. Направление потока слева направо и сверху вниз считается стандартным. В случаях, когда необходимо внести большую ясность в схему или поток имеет направление отличное от стандартного, на линиях используются стрелки, указывающие это направление.

В схемах следует избегать пересечения линий. Пересекающиеся линии не имеют логической связи между собой, поэтому изменения направления в точках пересечения не допускаются. Если две или более входящих линии объединяются в одну исходящую линию, то место объединения линий смещается.

Количество входящих линий не ограничено, выходящая линия из блока должна быть одна, за исключением логического блока.

Основными элементами блок-схем являются:

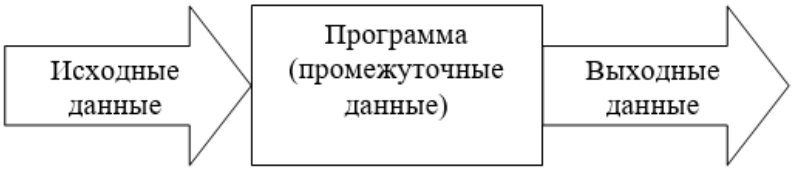
	- начало (конец) алгоритма
	- блок ввода-вывода данных
	- блок вычислений
	- логический блок, в котором направление потока информации выбирается в зависимости от некоторого условия
	- процесс пользователя (подпрограмма)
	- блок модификации, в котором функция выполняет действия, изменяющие пункты (например, заголовок цикла)
	- соединитель, используется для указания связи между потоками информации в пределах одного листа
	-межстраничный соединитель, т.е. указание связи между информацией на разных листах

Тема 1.3. Данные и их типы

Алгоритм, реализующий решение задачи, всегда работает с данными.

Данные – это любая информация, представленная в формализованном виде и пригодная для обработки алгоритмом.

По отношению к программе данные делятся на исходные, промежуточные и выходные.



Данные, известные перед выполнением алгоритма, являются начальными, **исходными данными**. Данные, используемые в процессе выполнения программы, являются **промежуточными данным***и*. Результат решения задачи – это конечные, **выходные данные**.

Данные делятся на переменные и константы.

Переменные – это такие данные, значения которых могут изменяться в процессе выполнения алгоритма.

Константы – это данные, значения которых не меняются в процессе выполнения алгоритма.

Любая величина имеет *3 основные свойства*:

1) **имя**, которое задается идентификатором, представляющим собой последовательность букв и цифр, начинающихся с буквы;
2) **значение**;
3) **тип данных** – это такая характеристика данных, которая задает множество допустимых значений и определяет множество операций, которые можно к этим данным применить.

Типы данных делят на 2 группы:

1)Простые (скалярные) типы – содержат одно единственное значение. К ним относятся:

1.**целый тип** – определяет подмножество допустимых значений из множества целых чисел (например: 23, -12);

2. **вещественный тип** - определяет подмножество допустимых значений из множества целых и дробных чисел в некотором диапазоне (например: 2,5; -0,01; $3{,}6\times10^{9}$);

3. **логический тип** – переменная принимает только два значения: истина (true) и ложь (false);

4. **символьный тип** – любые символы компьютерного алфавита (например: ‘a’, ‘5’, ‘+’).

2)Структурированные типы - описывают наборы однотипных или разнотипных данных (т.е. содержат несколько значений), с которыми алгоритм должен работать как с одной именованной переменной. К ним относятся: массивы, строки, множества и т.д.

Тема 1.4. Основные этапы решения задач

В настоящее время на ЭВМ решают самые разнообразные задачи. В каждом случае ЭВМ выполняет какую-то программу. Некоторые из

программ требуют от пользователя специальных знаний и высокой квалификации. Несмотря на бесконечное разнообразие программ, в самом процессе их изготовления можно выделить несколько этапов решения задачи на ЭВМ:

1) Постановка задачи

Под **постановкой задачи** понимают математическую или иную строгую формулировку решаемой задачи. Этот этап включает определение целей создаваемой программы и определение ограничений, налагаемых на программу. При постановке задачи должны быть определены требования:

• ко времени решения поставленной задачи;
• объему необходимых ресурсов, например, оперативной памяти;
• точности достигаемого результата.

2)Проектирование программы (формализация задачи)

Если задача вычислительная, то на этом этапе следует выбрать метод расчета, если разрабатывается компьютерная игра, должен быть определен ее сценарий. В любом случае следует выбрать или создать некую формальную модель, которая, в конечном счете, реализуется в будущей программе. На этапе проектирования определяют вид данных, с которыми будет работать программа, основные части, из которых программа будет состоять и характер связей между этими частями.

3) Разработка алгоритма

На этом этапе следует разработать детали проекта программы. Детализацию необходимо довести до той степени, когда кодирование деталей программы (перевод их на алгоритмический язык) станет тривиальным. Возможно, детализация потребует нескольких стадий, от крупных блоков к все более мелким, и в результате должно получиться то, что называется алгоритмом решения задачи.

4) Написание программы на языке программирования (кодирование)

После того как алгоритм разработан, его записывают на алгоритмическом языке, и этот процесс называют *кодированием* алгоритма. Для выполнения данного этапа необходимо знать хотя бы один из многих существующих языков программирования, а лучше знать несколько, чтобы выбрать наиболее подходящий для решаемой задачи.

5) Отладка и тестирование программы

Целью данного этапа является поиск и устранение ошибок в программе. Ошибки бывают синтаксические (нарушение грамматики алгоритмического языка) и смысловые (искажение самого алгоритма решения задачи).

6) Получение решения и анализ результатов

После проверки программы и устранении всех ошибок получают решение поставленной задачи, которое необходимо проанализировать. Если речь идет о моделировании какого-то природного процесса, то следует сравнить полученные с помощью компьютера результаты и результаты наблюдений. Они могут отличаться. В этом случае может

потребоваться возврат на один из предыдущих этапов для устранения причин несоответствия результатов.

Если же удалось разработать полезную программу, то работа над ней не заканчивается этапом тестирования, а переходит в фазу сопровождения. Программа живет, приобретает новые функции, совершенствует старые, избавляется от последних ошибок и, наконец, умирает, уступив натиску более новых и совершенных программ.

Таким образом, специалист должен обладать следующими знаниями и навыками:

1) уметь строить алгоритм;
2) знать языки программирования;
3)уметь работать в соответствующей системе программирования.

Тема 1.5. Основные структуры алгоритмов

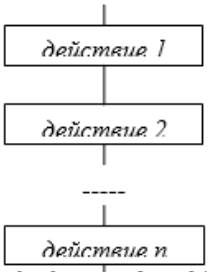
Основные структуры алгоритмов (ОСА) – это определенный набор блоков и стандартных способов их соединения для выполнения типичных последовательностей действий.

ОСА используются при структурном подходе к разработке алгоритмов и программ, предполагающем использование нескольких основных структур, комбинация которых дает все многообразие алгоритмов и программ.

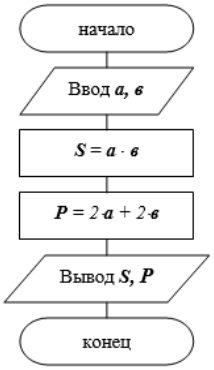
К основным алгоритмическим структурам относятся **линейные**, **разветвляющиеся** и **циклические** структуры.

1. Линейные алгоритмы

Линейными называются алгоритмы, в которых действия осуществляются последовательно друг за другом.



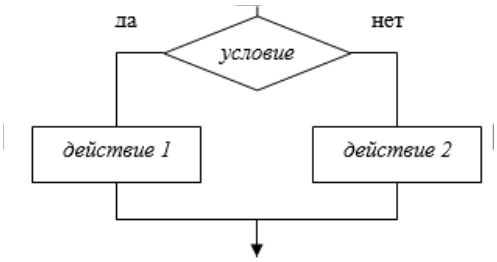
Пример: Разработать блок-схему алгоритма вычисления площади и периметра прямоугольника по двум заданным сторонам а и в.



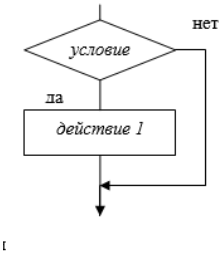
2. Разветвляющиеся алгоритмы

Разветвляющимся называется алгоритм, в котором действие выполняется по одной из возможных ветвей решения задачи, в зависимости от выполнения условий.

В качестве условия в разветвляющемся алгоритме может быть использовано любое понятное исполнителю утверждение, выраженное как словами, так и формулой. Оно может соблюдаться (быть истинным) или не соблюдаться (быть ложным). Таким образом, алгоритм ветвления состоит из условия и двух последовательных действий.

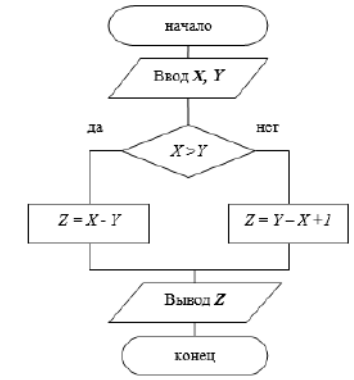


или



Пример: Разработать блок-схему алгоритма вычисления Z , если даны два действительных числа x и y .

$$Z = \begin{cases} X - Y, & \text{если } X > Y \\ Y - X + 1, & \text{если } X <= Y \end{cases}$$



3. Циклические алгоритмы

Циклическим называется алгоритм, в котором некоторая часть операций выполняется многократно.

Цикл - последовательность действий, выполняющихся многократно, каждый раз при новых значениях параметра.

Для организации цикла необходимо:

1)задать перед циклом начальное значение переменной, изменяющейся в цикле;

2)изменять переменную перед каждым новым повторением цикла;

3)проверять условие окончания или повторения цикла;

4)управлять циклом, т.е. переходить к его началу, если он незакончен, или выходить из него по окончании.

Последние три функции выполняются многократно.

Переменная, изменяющаяся в цикле, называется параметром.

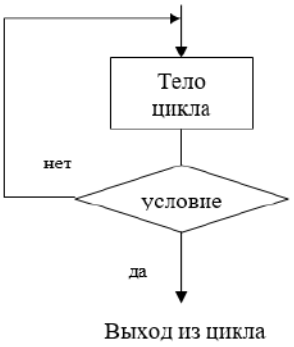
В цикл входят в качестве базовых следующие структуры: блок проверки условия и блок, называемый телом цикла.

В зависимости от способа организации числа повторений различают 3 типа циклов:
цикл с предварительным условием (цикл-ПОКА);
цикл с последующим условием (цикл-ДО);
цикл с параметром (цикл со счетчиком).

Цикл с предварительным условием имеет следующий вид:

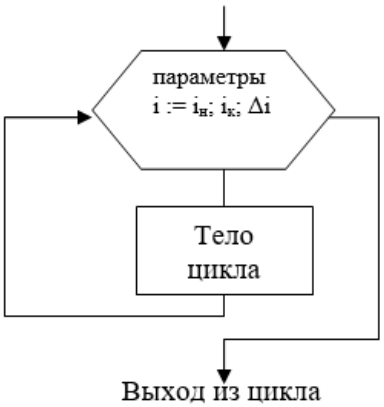
Если условие истинно, то тело цикла выполняется и управление передается снова на вычисление условия, если же условие ложно, то тело цикла не выполняется и происходит выход из цикла.

Цикл с последующим условием имеет следующий вид:



Если условие ложно, то вновь выполняются операторы тела цикла, если же условие истинно, то цикл заканчивается.

Цикл с параметром имеет следующий вид:

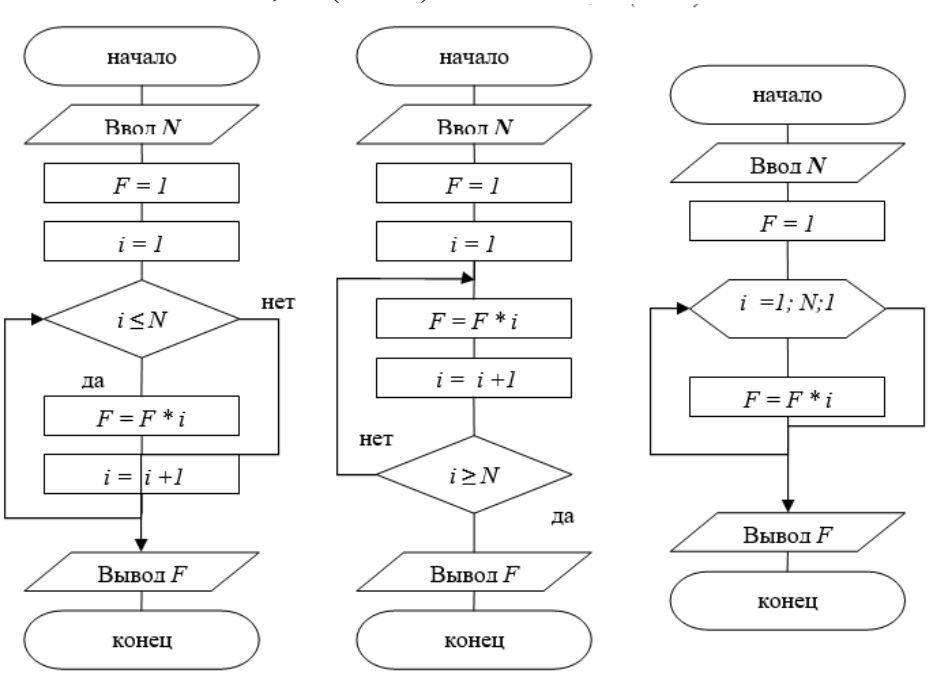


Циклические алгоритмы, в которых тело цикла выполняется заданное число раз, реализуются с помощью цикла с параметром.
В этом случае предусматривается повторное выполнение тела цикла с одновременным изменением по правилу арифметической прогрессии значения, присваиваемого параметру цикла.

При вычислении конечной суммы в циклическом алгоритме предварительно необходимо начальную сумму приравнять нулю (S = 0), а при вычислении конечного произведения – начальное произведение приравнять единице (P = 1).

Пример: Разработать блок-схему алгоритма вычисления факториала (F) натурального числа N. Факториал числа (!) – это произведение всех натуральных чисел от 1 до N.

$$N! = 1*2*3*\ldots* N \quad , \quad (0! = 1)$$



а) цикл с предварительным условием

б) цикл с последующим условием

в) цикл с параметром

Тема 1.6. Язык программирования

Язык программирования - формальная знаковая система, предназначенная для записи компьютерных программ.

Со времени создания первых программируемых машин человечество придумало более 8000 языков программирования, каждый год их число увеличивается. Профессиональные программисты, как правило, владеют несколькими языками программирования.

От естественных языков язык программирования отличается тем, что предназначен для взаимодействия человека с ЭВМ, а не для общения людей между собой.

Язык программирования определяет набор правил:

лексических (словарный запас языка);

синтаксических (набор правил, описывающий комбинации символов алфавита, считающиеся правильно структурированной программой (документом) или ее фрагментом);

семантических (начальное смысловое значение операторов, основных конструкций языка и т.п.).

Которые задают внешний вид программы и действия, которые выполнит исполнитель под ее управлением.

Как правило, язык программирования существует в нескольких, но существенно отличающихся видах:

стандарта языка: набора спецификаций, определяющих его синтаксис и семантику; стандарт языка может исторически развиваться;

воплощений (реализаций) стандарта: собственно программных средств, обеспечивающих работу согласно тому или иному варианту стандарта языка.

Например, язык C++ имеет официальный стандарт (ISO/IEC 14882:2014) и различные реализации: GNU C++, Microsoft Visual C++ и др.

Машинный код

Первым языком программирования можно считать **Машинный код** где каждая инструкция выполняет определенное действие:

операция с данными (например, сложение или копирование машинного слова в регистре или в памяти);

переход к другому участку кода (изменение порядка исполнения; переход может быть безусловным или условным, зависящим от результатов предыдущих инструкций)

при этом представляя несколько 0 и 1 в более удобном виде.

Любая исполнимая программа состоит из последовательности таких атомарных машинных операций. При исполнении очередной инструкции специальный блок процессора — декодер — транслирует (декодирует) ее в последовательность элементарных операций, понимаемых конкретными исполнительными устройствами.

Языки ассемблера

Попытка сделать машинные команды более человекочитаемыми привели к созданию языков ассемблера (1950-е гг.). Команды языков данного вида как правило один в один соответствуют командам процессора и, фактически, представляют собой удобную символьную форму записи (мнемокоды) команд и их аргументов.

Код на языке ассемблера в конечном итоге переводится в машинный код, используя специальную программу-преобразователь (транслятор), несомненным плюсом языков ассемблера по сегодняшний день является крайне эффективная работа программы (быстрый и компактный код).

В то же время поддержка подобной программы крайне трудоемка, код пишется для конкретного устройства, и ее перенос на другие устройства влечет за собой практически полное переписывание, что послужило причиной появления языков высокого уровня.

азываемую ассемблером.

40-е гг.: первый язык высокого уровня

Машинный код и языки ассемблера традиционно относят к языкам низкого уровня - программирование на них осуществляется либо непосредственно в машинных кодах используемого реального или виртуального процессора, либо приближено к этому.

В скором времени начинает преобладать идея, что человек должен писать программы на более легком для него языке, чем машинные коды, а трудоемкий перевод в машинный код переложить на другую программу (транслятор), при этом транслятор также должен уметь в какой-либо степени делать перевод на машинный код различных устройств. Данная идея лежит в основе всех современных языков программирования.

Языки программирования, которые имеют достаточный уровень абстракции - оперируют сложными структурами данных и умеют выполнять над ними комплексные операции (что в машинном коде повлекло бы длинные и сложные цепочки кода), относят к языкам высокого уровня. Примеры таких языков - C, Pascal, Python, Java и др.

В общем случае целью создания новых языков программирования было упрощение написание программ и повышение уровня абстракции (от команд конкретной машины к языку человека).

Исторически первым спроектированным языком программирования высокого уровня был Планкалкюль (нем. Plankalkül - исчисление планов), созданный немецким инженером Конрадом Цузе в 1943-45 году и впервые опубликованный в 1948 году.

Планкалкюль обладал многими возможностями, которыми обладают современные языки, однако, работа в отрыве от других специалистов Европы и США привела к тому, что лишь незначительная часть его работы стала известной, а сама разработка осталась теоретической. Полностью работа Цузе была издана лишь в 1972 году, а первая реализация языка Планкалкюль (для современных компьютеров) была создана в Свободном университете Берлина лишь в 2000 году, через пять лет после смерти Конрада Цузе.

50-е гг.: сфера науки и бизнеса

В 50-е гг. в приоритете остаются задачи научного характера (математика, физика, искусственный интеллект). Программы на языках данного периода представляют собой последовательность команд, которые должен выполнить компьютер.

Fortran

В период 1954-1957 гг. американским ученым в области информатики Джоном Бэкусом совместно с компанией IBM разрабатывается Fortran (англ. FORmula TRANslator) - первый язык программирования общего назначения применяемый до сих пор (он совершенствуется и вбирает в себя различные нововведения, а последняя версия датируется 2010 г.), существует большое количество написанных на нем программ и библиотек подпрограмм.

Lisp

В 1958 г. американский информатик Джон Маккарти изобретает язык Lisp (англ. LISt Processing language) - первый язык программирования,

который до сих пор является одним из основных средств моделирования различных аспектов искусственного интеллекта.

Нововведениями Лиспа являлись:

представление программ и данных системами линейных списков символов;

автоматическое управление памятью и сборка мусора (язык самостоятельно освобождал неиспользуемую занятую ранее память компьютера).

COBOL

В 1959 г. под руководством американской ученой и контр-адмирала флота США Грейс Хоппер создается язык COBOL (англ. COmmon Business Oriented Language), предназначенный, в первую очередь, для разработки бизнес-приложений.

Начиная карьеру как программист компьютера Марк I, Хоппер также вошла в историю как:

разработчик первого компилятора (транслятора, который преобразует высокоуровневый код в машинный код целевой машины) для компьютерного языка программирования;

сторонник и разработчик концепции машинно-независимых языков программирования;

популяризатор термина «отладка» (исправление сбоев в работе программы).

Кобол - стандартизованный язык практически с момента своего рождения, используемый и сегодня - еще в 2006 году Кобол считался языком программирования, на котором было написано больше всего строк кода.

Algol

В 1958—1960 гг. появляется Algol (англ. ALGOrithmic Language) - язык, повлиявший практически на все языки следующего десятилетия (Pascal, C и другие). Алгол был популярен в Европе, в том числе в СССР, в то время как сравнимый с ним язык Фортран был распространен в США и Канаде. Алгол является предком довольно большого числа современных языков программирования (от Си и далее).

К нововведениям Алгола стоит отнести:

вложенные блоки: код мог быть сгруппирован в блоки (например, условие внутри условия) без необходимости вынесения кода в отдельную процедуру;

понятие области видимости: блок программы мог иметь свои закрытые переменные, процедуры и функции, невидимые для внешнего кода.

60-70-е гг.: универсальность

С повышением доступности компьютеров (например, в 1964 г. появляется популярная модель IBM 360) и появлением возможности работы в режиме разделения времени (time-sharing), доступ к компьютеру получили не только крупные научно-исследовательские центры, но и предприятия, образовательные учреждения, в том числе учащиеся и

специалисты, не являющиеся подготовленными программистами, но нуждающиеся в решении на компьютере своих задач.

В связи с этим языки программирования продолжают совершенствоваться. В данный период также появляются основные парадигмы программирования, в рамках которых развиваются языки.

PL/I

В 1964 г. IBM разрабатывает язык PL/I (англ. Programming Language I), предназначавшийся для научных, инженерных и бизнес-ориентированных вычислений. В это время научные и бухгалтерские программы не только использовали разные компьютеры, но еще и писались на разных языках: научные — на Фортране, бухгалтерские — в основном на Коболе. Целью ПЛ/1 было создание языка, подходящего для обоих типов приложений.

Язык ввел в употребление множество нововведений, взятых на вооружение другими языками позже:

блочная структура;

большой набор встроенных типов данных;

мощный механизм итераций;

обработка исключительных ситуаций;

макросы и отладка.

Несмотря на заявленные характеристики и мощные возможности, PL/I не добился такого же успеха как Fortran или COBOL в виду сложной и неоптимальной для того времени реализации.

Basic

К 1964 г. появляется язык Basic (англ. Beginner’s All-purpose Symbolic Instruction Code — универсальный код символических инструкций для начинающих), создававшийся как инструмент, основной задачей которого было предоставить студентам-непрограммистам возможность после минимального обучения самостоятельно писать простейшие программы для собственных нужд, чтобы не отвлекать на эту работу специалистов. При этом подразумевалось, что на Бейсике должна быть возможность писать относительно эффективные программы, хоть и без знания принципов аппаратного обеспечения.

Несмотря на свою простоту Бейсик завоевал популярность и даже сегодня остается востребованным и используется в различных диалектах, например, в качестве встроенного языка текстового процессора Microsoft Word (VBA), или средства визуального программирования Visual Basic, а также других вариантах.

Pascal

В 1970 г. известный швейцарский ученый в области информатики Никлаус Вирт создает язык Pascal - один из наиболее известных языков программирования. Основной плюс языка как в то время, так и сейчас - баланс простоты/возможностей и привитие хорошего стиля программирования - до сих пор оставляет его популярным для обучения программированию и использования в ряде коммерческих приложений.

Несмотря на угасающую популярность, язык непрерывно развивается: сегодня существует множество диалектов, поддерживающих как визуальное и объектное программирование (Delphi или Lazarus/FreePascal), так и являющихся прямыми «улучшенными» потомками (Modula, Oberon и др.).

Prolog

В 1972 г. появляется первый язык логического программирования Prolog, основанный на теории и аппарате математической логики.

Smalltalk

В 1972 г. в научно-исследовательской компании Xerox PARC американский ученый в области теории вычислительных систем Алан Кэй с коллегами разрабатывает первый объектно-ориентированный язык общего назначения Smalltalk.

Smalltalk привнес следующие нововведения:

все — объекты: строки, целые числа, логические значения, определения классов, блоки кода и т.д.;

работа программы - изменение свойств объекта и вызов действий, которые они поддерживают;

динамическая типизация: типы переменных не указываются в программе, а является ли операция правильной, определяет объект-получатель;

автоматическая сборка мусора;

выполнение кода в виртуальной машине: программы Smalltalk обычно компилируются в байткоды и выполняются виртуальной машиной, что позволяет выполнять их на любом оборудовании, для которого существует виртуальная машина (то, что сегодня делает Java).

C

В период с 1969 по 1973 г. сотрудники американской телекоммуникационной компании Bell Labs Деннис Ритчи и Кен Томпсон разрабатывают язык C - язык общего назначения, изначально предназначенный для реализации операционной системы UNIX.

Язык Си уникален с той точки зрения, что именно он стал первым языком высокого уровня, всерьез потеснившим ассемблер в разработке системного программного обеспечения (согласно дизайну языка Си, его конструкции близко сопоставляются типичным машинным инструкциям). Он остается языком, реализованным на максимальном количестве аппаратных платформ, и одним из самых популярных языков программирования, особенно в мире свободного программного обеспечения. Си послужил основой для многих современных языков программирования, в том числе визуальной разработки, а также продолжает развиваться и сам (последний стандарт датируется 2011 г.).

Ada

Ada - язык, разрабатываемый в течение 70-х гг. Министерством обороны США с целью разработать единый язык программирования для встроенных систем (бортовых систем управления военными объектами -

кораблями, самолетами, танками, ракетами, снарядами и т.п.), функционирующих в режиме реального времени.

Синтаксис Ады унаследован от языков типа Algol или Паскаль, но расширен, а также сделан более строгим и логичным. Для удовлетворения требованиям надежности язык построен таким образом, чтобы как можно большее количество ошибок обнаруживалось на этапе компиляции. Кроме того, одним из требований при разработке языка была максимально легкая читаемость текстов программ, даже в ущерб легкости написания. Язык используется в том числе отечественными военными (на нем, например, написана станция документальной связи МО РФ), однако в целом малопопулярен.

80-е гг.: консолидация идей и модульность

80-е гг. символизируются консолидацией идей, новые языки пытаются совместить изобретенные ранее парадигмы. Ближе к концу 80-х гг. начинают появляться скриптовые языки (языки, как правило, предназначенные для автоматизации каких-либо действий - быстрого поиска в тексте, автоматизации задач операционной системы и т.д.).

При проектировании языков программирования возросло внимание к программированию масштабируемых систем с использованием модулей (модуль - функционально законченный фрагмент программы, который можно использовать в программе). Удобство использования модульной архитектуры заключается в возможности обновления (замены) модуля, без необходимости изменения остальной системы.

Дальнейшее развитие получают идеи обобщенного программирования, заключающиеся в таком описании данных и алгоритмов, которое можно применять к различным типам данных, не меняя само это описание. В том или ином виде поддерживается разными языками программирования (до появления данной идеи, например, для сортировки целых чисел и дробных необходимо было записать 2 разные процедуры в виду различия типов).

C++

В 1983 г. программист компании Bell Бьерн Страуструп представляет язык C++ - первый язык, совместивший черты высокоуровневого языка (объектно-ориентированного, в частности), и низкоуровнего.

Objective-C

В то же время 1983 г. в компании Apple под руководством Брэда Кокса разрабатывается собственный Си с классами – Objective-C, построенный на основе языка Си и идей Smalltalk.

Имеет графическую оболочку и позволяет создавать приложения с графическим интерфейсом пользователя. При этом язык используется и поддерживается только в продуктах Apple, поэтому не получил такого широкого распространения как C++.

Eiffel

В 1986 г. французский ученый в области информатики Бертран Мейер создает язык Eiffel (Эйфель) — объектно-ориентированный язык программирования с алгола подобным синтаксисом, в котором впервые был реализован метод контрактного программирования.

Метод контрактного программирования предполагает, что проектировщик должен определить формальные, точные и верифицируемые спецификации интерфейсов для компонентов системы. Данные спецификации называются «контрактами» в соответствии с концептуальной метафорой условий и ответственности в гражданско-правовых договорах.
Eiffel используется в серьезных разработках, от которых зависят жизни людей: в аэрокосмической отрасли, в банковско-финансовой сфере и др. Несмотря на это язык слабо распространен, т.к. компиляторы с Eiffel от автора языка дороги и не так распространены, как компиляторы C/C++, что, в свое время, и ограничило распространение этого языка, а свободных/бесплатных альтернатив долгое время не было.

Perl

Ларри Уолл, лингвист по образованию, в 1987 г. создает язык Perl (*англ.* Practical Extraction and Report Language - «практический язык для извлечения данных и составления отчетов») - язык программирования общего назначения, который был первоначально создан для манипуляций с текстом, но на данный момент используется для выполнения широкого спектра задач, включая системное администрирование, веб-разработку, сетевое программирование, игры, биоинформатику, разработку графических пользовательских интерфейсов. Язык можно охарактеризовать скорее как практичный (легкость в использовании, эффективность, полнота), чем красивый (элегантность, минималистичность).

Tcl

В 1988 г. американский ученый в области информатики Джон Оустерхаут создает язык Tcl (*англ.* Tool Command Language - «командный язык инструментов») - один из первых скриптовых языков высокого уровня. Основная область применения - быстрое прототипирование, создание графических интерфейсов для консольных программ (пакетов программ), встраивание в прикладные программы, тестирование. Также Tcl применяется в веб-разработке.

Интернет-эпоха

90-е годы характеризовались распространением сети Интернет и персональных компьютеров, а также увеличением количества данных, что послужило очередным толчком для дальнейшего развития языков программирования. Многие языки, созданные в это время, являются свободными (пользователи имеют права на неограниченную установку, запуск, свободное использование, изучение, распространение и изменение (совершенствование), а также распространение копий и результатов изменения), что также явилось фактором широкого распространения ряда языков.
Можно выделить несколько тенденций и идей развития в это время:

- продолжение комбинации идей предыдущих десятилетий и распространение функциональных языков;
- повышение продуктивности разработки;
- продолжение развития скриптовых (динамических) языков;

- графический интерфейс пользователя (ГПИ, *англ.* Graphical User Interface, GUI).

Желание увеличить производительность работы программиста в совокупности с сохранением проверенных временем идей привело к появлению новых языков высокого уровня, а также сред быстрой разработки (*англ.* Rapid Application Development - RAD), включающих кроме самого языка и транслятора интегрированную среду разработки (*англ.* Integrated Development Environment - IDE). Все такие языки программирования были объектно-ориентированными, а в ряде случаев IDE также позволяла выполнять графическое проектирование приложений.

Python

В 1991 г. нидерландский программист Гвидо ван Россум представляет Python — высокоуровневый мультипарадигменный язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Синтаксис ядра Python минималистичен, в то же время стандартная библиотека включает большой объём полезных функций.

Python портирован и работает почти на всех известных платформах — от КПК до мейнфреймов. Язык используется в различных качествах: как основной язык программирования или для создания расширений и интеграции приложений. На Python реализовано большое количество проектов, также он активно используется для создания прототипов будущих программ. Python используется в основном в качестве языка веб-программирования, обработки данных и научных задач.

Ruby

Спустя 2 года, в 1993 г., японский разработчик Юкихиро Мацумото создает язык Ruby - высокоуровневый язык программирования для быстрого и удобного объектно-ориентированного программирования, который, по словам автора, более мощный, чем Perl, и более объектно-ориентированный, чем Python.

Ruby имеет немало оригинальных решений, редко или вообще не встречающихся в распространенных языках программирования - расширенная работа с массивами, классами и т.д.

Основные сферы применения приблизительно такие же, как и у Python.

Lua

Еще один скриптовый язык программирования Lua в 1993 г. в Бразилии разрабатывает преподаватель университета Роберто Иерусалимский совместно с коллегами.

Язык широко используется для создания тиражируемого программного обеспечения (например, на нем написан графический интерфейс пакета Adobe Lightroom). Также получил известность как язык программирования уровней и расширений во многих играх (в том числе World of Warcraft) из-за удобства встраивания, скорости исполнения кода и легкости обучения.

R

В 1993 г. также создается R - язык программирования для статистической обработки данных и работы с графикой. R широко используется как программное обеспечение для анализа данных и фактически стал стандартом для статистических программ, имеет собственные IDE.

JavaScript

В 1995 г. набирает популярность язык JavaScript как встраиваемый язык для программного доступа к объектам приложений. Наиболее широко применяется в браузерах как язык сценариев для придания иPHP

В 1995 г. появляется PHP (*англ.* PHP: Hypertext Preprocessor — «PHP: препроцессор гипертекста») — скриптовый язык общего назначения, интенсивно применяемый для разработки веб-приложений. В настоящее время поддерживается подавляющим большинством хостинг-провайдеров и является одним из лидеров среди языков, применяющихся для создания динамических веб-сайтов.

нтерактивности веб-страницам.

Java

Также в 1995 г. появляется язык Java - язык программирования, разработанный компанией Sun Microsystems (в последующем приобретенной компанией Oracle). Приложения Java обычно транслируются в специальный байт-код, поэтому они могут работать на любой компьютерной архитектуре, с помощью виртуальной Java-машины. Достоинством подобного способа выполнения программ является полная независимость байт-кода от операционной системы и оборудования, что позволяет выполнять Java-приложения на любом устройстве, для которого существует соответствующая виртуальная машина. Другой важной особенностью технологии Java является гибкая система безопасности, в рамках которой исполнение программы полностью контролируется виртуальной машиной - любые операции, которые превышают установленные полномочия программы (например, попытка несанкционированного доступа к данным или соединения с другим компьютером), вызывают немедленное прерывание.

Java-технологии используются в огромном количестве устройств (более 3 млрд.), используется как в качестве разработки настольных и мобильных приложений, так и веб-приложений; кроме того, существует множество сред визуальной разработки, поддерживающих язык Java.

Тема 1.7. Классификация языков программирования

Языки программирования классифицируются по различным категориям, рассмотрим их основные группы.

Низкоуровневые и высокоуровневые языки

Машинный код и языки ассемблера традиционно относят к языкам низкого уровня - программирование на них осуществляется либо непосредственно в машинных кодах используемого реального или виртуального процессора, либо приближено к этому.

Языки программирования, которые имеют достаточный уровень абстракции - оперируют сложными структурами данных и умеют выполнять над ними комплексные операции (что в машинном коде повлекло бы длинные и сложные цепочки кода), относят к языкам высокого уровня. Примеры таких языков - C, Pascal, Python, Java и др.

Трансляторы: компиляторы, интерпретаторы и гибриды

Как и в случае с языками ассемблера, любой язык программирования должен пройти процесс перевода на машинный код для того, чтобы компьютер понял какие инструкции ему дают в рамках того, что он умеет.

Программа, предназначенная для перевода кода с языка программирования в машинный код называется **транслятором**. На сегодняшний день выделяется 3 типа трансляторов: компиляторы, интерпретаторы и гибриды.

Компиляторы однократно осуществляют перевод всего кода в машинный код. Результатом процесса компиляции является исполняемый модуль (например, EXE-файл для ОС Windows).

Интерпретаторы анализируют и тут же выполняют программу покомандно, по мере поступления ее исходного кода на вход интерпретатора.

Каждый из вариантов имеет свои плюсы и минусы (Таблица 1)

№	Характеристика	Компилятор	Интерпретатор
1	Быстродействие	(+) Обеспечивают максимальную производительность	(-) Дополнительный слой интерпретатора замедляет выполнение программы и может требовать больше ресурсов
2	Переносимость	(-)Исполняемый файл привязан к платформе компиляции	(+) Программа может быть легко перенесена на другую платформу, для которой имеется интерпретатор

Таблица 1. - Преимущества и недостатки компиляторов и интерпретаторов

Гибриды пытаются совместить плюсы компиляторов и интерпретаторов. Наиболее известные представители - язык Java и платформа .NET Framework.

Практически все современные языки относительно «размывают» границы данной классификации и в той или иной степени являются типизация гибридными.

Языки высокого уровня работают не с битами и байтами, а с более абстрактными структурами данных - типами, поэтому эти языки принято считать типизированными.

Выделяют несколько видов типизации :

1. Явная / неявная.
2. Статическая / динамическая.
3. Сильная (строгая) / слабая (нестрогая).

Явная / неявная

1. Явная: тип новых переменных, функций и их аргументов необходимо задавать явно.

Примеры: C++, Pascal, C#.

2. Неявная: языки с неявной типизацией перекладывают эту задачу на компилятор / интерпретатор

Статическая / динамическая

1. Статическая: конечные типы переменных и функций устанавливаются на этапе компиляции.

Примеры: C, Java, C#.

2. Динамическая: все типы выясняются во время выполнения программы.

Примеры: Python, JavaScript.

Понятия «статический» или «динамический» язык программирования подразумевают данный вид типизации.

Сильная / слабая

1. Сильная (строгая): не позволяет смешивать в выражениях различные типы и не выполняет автоматические неявные преобразования

Примеры: Java, Python.

2. Слабая (нестрогая): множество неявных преобразований происходит автоматически, даже если может произойти потеря точности или преобразование неоднозначно.

Примеры: C, JavaScript, Visual Basic, PHP.

Все виды типизации пересекаются, например, язык C имеет [явную, статическую, слабую] типизацию, а язык Python — [неявную, динамическую, сильную].

Тема 1.8. Методы и принципы программирования

Методы программирования

При разработке программ используют ***3 метода*** их построения:

1)Структурное программирование – описывается каждое действие алгоритма для достижения конечного результата, используя процедурный стиль программирования и последовательную декомпозицию алгоритма решения задачи сверху вниз;

2)Модульное программирование – проектирование новой программной системы на базе разработанных и отлаженных ранее модулей. Этот метод многократного использования разработанного программного обеспечения.

3)Объектно-ориентированное программирование – описывает программные системы в виде взаимодействия объектов.

Объект – это совокупность данных и действий над ними. Каждый объект обладает свойствами. **Свойства** – это характеристики состояния объекта.

Взаимодействие с объектом происходит через интерфейс, использующий *визуальные компоненты*.

Таким образом, структурное программирование легло в основу всех методов программирования.

В современных условиях широко развивается объектно-ориентированное программирование, которое имеет следующие достоинства: упрощение процесса проектирования программных систем, легкость их сопровождения и модификации, минимизирование времени разработки за счет многократного использования готовых модулей.

Этот метод используется в современных *средах разработки приложений*. При формировании общего вида главного окна при выполнении приложения и способов управления работой приложения для каждого компонента определяют внешний вид, размеры, способ и место размещения в области окна приложения.

Компоненты разбиты на две группы:

1)Визуальные компоненты (элементы управления – кнопки, списки, переключатели, надписи) имеющие фиксированное местоположение и размеры. Они подразделяются на «оконные» (становятся активными для взаимодействия с пользователем) и «неоконные» (графические).

2)Невизуальные компоненты не имеют фиксированного местоположения и размеров (диалоговые окна открытия и сохранения файлов, таблицы баз данных).

При разработке процедур обработки событий необходимо запрограммировать реакцию на всевозможные изменения состояния объектов.

Тема 1.9. Виды программного обеспечения. Общие принципы разработки ПО

Программное обеспечение (ПО) – совокупность программ, которые могут выполняться вычислительной системой.

Программное обеспечение различается по назначению, выполняемым функциям, формам реализации.

Программное обеспечение разделяется на ***3 вида***:

1)Системное (*базовое* – операционные системы; *сервисное* – программы диагностики работоспособности компьютера, программы обслуживания дисков, программы архивирования данных, антивирусные программы);

2)Прикладное (текстовые редакторы, электронные таблицы, системы баз данных, графические редакторы, программы проектирования, обучающие программы и др.);

3)Инструментальное (интегрированные среды разработки приложений, системы программирования, средства создания информационных систем).

При разработке ПО вне зависимости от его вида используют **общие принципы**:

частотный принцип – выделение действий и данных по частоте использования;

принцип модульности – обособление составных частей ПО в отдельные модули по функциональному признаку;

принцип функциональной избирательности – при проектировании ПО, объем которого превышает объем оперативной памяти, часть ПО, которая постоянно используется, хранится в оперативной памяти, другая часть – на внешних запоминающих устройствах;

принцип генерируемости – осуществление настройки на конкретную конфигурацию технических средств, круг решаемых проблем, условия работы пользователя;

принцип функциональной избыточности – возможность проведения одной и той же работы (функции) различными средствами;

принцип «по умолчанию» - хранение в системе базовых описаний структур, модулей, конфигураций оборудования и данных, определяющих условия работы с ПО.

При создании и развитии ПО применяют также общесистемные принципы:

принцип включения – требования к созданию и функционированию ПО определяются включающей его в себя системы;

принцип системного единства – обеспечение целостности между подсистемами;

принцип развития – возможность наращивания и совершенствования компонентов ПО и связей между ними;

принцип комплексности – обеспечение связности обработки информации;

принцип информационного единства – использование единых терминов, символов, условных обозначений и способов представления;

принцип совместимости – обеспечение совместного функционирования всех подсистем ПО;

принцип инвариантности – подсистемы и компоненты ПО универсальны по отношению к обрабатываемой информации.

Жизненный цикл программного обеспечения

Решение задачи на ЭВМ проходит поэтапно и предусматривает возможность возврата на предыдущие этапы после анализа полученных результатов.

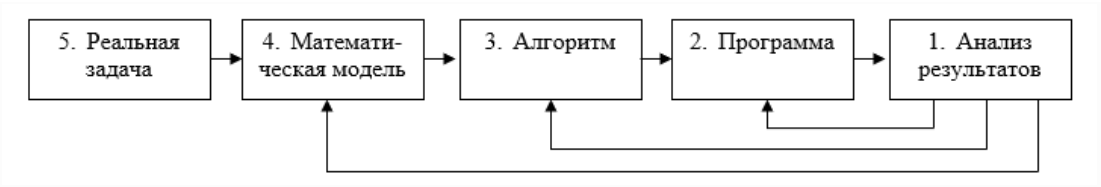


Рисунок 1. Технологическая цепочка решения задач на ЭВМ

Жизненный цикл ПО – это процесс от начала создания ПО до полного изъятия из эксплуатации.

Структура жизненного цикла содержит процессы, действия и задачи, которые должны быть выполнены во время создания ПО.

Жизненный цикл ПО состоит из следующих ***стадий***:

1) Определение требований и спецификаций – устанавливаются общие требования по надежности, технологичности, универсальности, эффективности, информационной согласованности, разрабатываются входные и промежуточные языки, формы выходной информации.

2)Проектирование ПО – формируется структура ПО, разрабатываются алгоритмы, устанавливается состав модулей и интерфейсы.

3)Программирование – проектные решения реализуются в виде программ.

4)Отладка ПО – проверка выполнения всех требований, всех структурных элементов системы, работоспособности ПО, выявление и исправление ошибок.

5)Сопровождение ПО – процесс координации всех элементов системы в соответствии с требованиями пользователя, внесения всех необходимых ему исправлений и изменений, дальнейшее совершенствование системы во время эксплуатации.

В процессе разработке ПО приходится осуществлять несколько итераций (последовательных приближений) к приемлемому варианту системы.

Глава 2. Язык программирования Python

Тема 2.1. Введение в язык программирования Python. Среда Python.

Язык Python является одним из самых простых в изучении и самых приятных в использовании из языков программирования, получивших широкое распространение. Программный код на языке Python легко читать и писать, и, будучи лаконичным, он не выглядит загадочным. Python - очень выразительный язык, позволяющий уместить приложение в меньшее количество строк, чем на это потребовалось бы в других языках, таких как C++ или Java.

Python - интерпретируемый, объектно-ориентированный высокоуровневый язык программирования с динамической семантикой. Это высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода.

Python— один из самых популярных языков программирования на сегодняшний день.

У Python имеется ряд очевидных преимуществ перед другими языками программирования:

- интерпретирующий характер: язык обрабатывается интерпретатором во время выполнения;
- интерактивность: вы можете напрямую взаимодействовать с интерпретатором при написании вашей программы;
- идеально подходит для начинающих программистов.
- универсальное использование – на этом языке можно писать разные типы приложений, потому вместе с его освоением открываются широкие возможности для применения этого языка;
- простота – изначально язык разрабатывался для упрощения работы с ним человека;
- популярность в среде программистов и востребованность на рынке труда – Python широко применяется в различных проектах;
- большое количество доступных библиотек расширяют возможности языка и делают его еще более универсальным;
- кроссплатформенность – один раз написанная программа будет работать на любой платформе, где есть интерпретатор языка;
- одним из важных плюсов языка является его качественная документация.

Python имеет огромное количество высококачественных уже готовых модулей, распространяемых бесплатно, которые вы можете использовать в любой части программы. В модуле уже реализованы многие нужные вам детали программы. Модули подключаются при помощи команды import, которая присутствует в начале каждого примера. Все широко используемые модули делятся на две основные части: модули стандартной библиотеки, поставляемые вместе с интерпретатором Python (эти модули «всегда с вами»), и внешние модули, для которых существуют средства установки.

Создание Python было начато Гвидо ван Россумом (Guido van Rossum) в 1991 году, когда он работал над распределенной ОС Амеба. Ему требовался расширяемый язык, который бы обеспечил поддержку системных вызовов. За основу были взяты ABC и Модула-3.

В качестве названия он выбрал Python в честь комедийных серий BBC «Летающий цирк Монти-Пайтона», а вовсе не по названию змеи. С тех пор Python развивался при поддержке тех организаций, в которых Гвидо работал. Особенно активно язык совершенствуется в настоящее время, когда над ним работает не только команда создателей, но и целое сообщество программистов со всего мира. И все-таки последнее слово о направлении развития языка остается за Гвидо ван Россумом.

Логотип (рис.2) создал брат автора, Юст ван Россум — программист и шрифтовой дизайнер. Он разработал как дизайн логотипа (две змеи), так и шрифт текста Flux Regular.

Рисунок 2. Логотип языка

На логотипе изображены две змеи, образующие квадрат с выпуклым центром, это часто вводит в заблуждение пользователей, вынуждая ассоциировать название языка с рептилией.

В настоящее время в русском языке для обозначения используют два варианта – «Питон» и «Пайтон».

Какие возможности дает Python:

- может использоваться на сервере для создания веб-приложений.
- может использоваться вместе с программным обеспечением для создания рабочих процессов.
- может подключаться к системам баз данных. С его помощью можно также читать и изменять файлы.
- может использоваться для обработки больших данных и выполнения сложных математических вычислений.
- может использоваться для быстрого прототипирования или для разработки готового программного обеспечения.

Среда Python

Язык программирования Python это инструмент для создания программ самого разнообразного назначения, доступный даже для новичков.

Python-код легко читается, а интерактивная оболочка позволяет вводить программы и сразу же получать результат. Благодаря понятному синтаксису на нем легко начать программировать.

Для программирования на языке Python необходимо установить бесплатную среду программирования у себя на компьютере, которую можно загрузить по адресу: https://www.python.org/downloads/.

В комплекте вместе с интерпретатором Python идет IDLE (интегрированная среда разработки). По своей сути она подобна интерпретатору, запущенному в интерактивном режиме с расширенным набором возможностей (подсветка синтаксиса, просмотр объектов, отладка и т.п.).

Интерпретатор - это программа, которая читает вашу программу и выполняет содержащиеся в ней инструкции.

Для запуска IDLE в Windows необходимо перейти в папку Python в меню “Пуск” и найти там ярлык с именем “IDLE (Python 3.X XX-bit)”.

Для запуска редактора программы (кода) следует выполнить команду File->New File или сочетание клавиш Ctrl+N.

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

Программа включает в себя:

- комментарии;

- команды;
- знаки пунктуации;
- идентификаторы;
- ключевые слова.

Для того чтобы записывать, сохранять и выполнять Python-команды, необходимо сделать следующие шаги:

1 Шаг. Запустить IDLE

Откроется окно консоли, в котором вы будете видеть результат вашей программы.

Python3.72
Shell

File Edit Shell Debug Options Window Help

Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 23:09:28) [MSC v.1916 64 bit (AMD64)] on win32
Type "'help", "copyright', "credits" or "license()" for more information.
>>>

2 Шаг. Создать новый файл

Для записи новой программы необходимо создать отдельный файл. Для этого в меню **File** выберите **New File**:

Python 3.7.2 Shell

File Edit Shell Debug Options

New File	Ctrl+N

Open… Ctrl+o
Open Module Alt+M

3 Шаг. Ввести программу

В открывшемся окне введите вашу программу, например:

```
print ('Salam! ')
```

4 Шаг. Сохранить программу

В меню **File** выберите **Save As**, задайте новое имя для вашего файла и нажмите **Save.** Мы рекомендуем вам предварительно создать папку My scripts и сохранять туда все свои программы.

5 Шаг. Запустить программу

В меню **Run** выберите **Run Module** (либо можно просто нажать быструю клавишу F5)

Готово! В самой консоли вы должны увидеть сообщение:

>>>
Salam!

Язык Python поддерживается всеми операционными системами (существуют версии для Linux, Windows, MacOS) и позволяет решать сложные математические задачи, создавать графические изображения, разрабатывать веб-сайты, работать с реляционными базами данных. Он

используется для решения большого количества как научных, так и бизнес-задач. Программы могут разрабатываться в консольном режиме (такие программы имеют расширение .py) и с графическим интерфейсом (программы имеют расширение .pyw). Программа на языке Python — это обычный текстовый файл, инструкции (команды) которого выполняются интерпретатором для каждой строки. Программу на языке Python можно создавать и редактировать с помощью любого редактора, например, Notepad ++, Eclipse, Nano и др. Язык Python отличает скорость и простота скриптов. Все данные языка, в том числе простые типы данных (числа, строки) являются объектами. В переменной хранится не сам объект, а ссылка на него, то есть адрес ячейки памяти, в которой хранится объект.

Тема 2.2. Основные элементы языка Python

Любая конструкция языка программирования начинается с алфавита. Из символов алфавита создаются лексемы (token). Лексема — это минимальная единица языка, которая имеет определенное самостоятельное значение.

Различают следующие виды лексем:

- ключевые (зарезервированные) слова (keywords); - идентификаторы (identifiers);

- литералы (константы);

- операции;

-знаки препинания.

В языке Python используется несколько десятков ключевых (зарезервированных) слов (например, int, list, input, print, float и др.). Идентификаторы (имена) используются для обозначения переменных, функций, которые создает программист, и других объектов.

Имена переменных могут быть любыми.

Однако есть несколько общих правил их написания:

1) Желательно давать переменным осмысленные имена, говорящие о назначении данных, на которые они ссылаются.

2) Имя переменной не должно совпадать с командами языка (зарезервированными ключевыми словами).

3) Имя переменной должно начинаться с буквы или символа подчеркивания _, но не с цифры.

4) Имя переменной не должно содержать пробелы.

Имена переменных используются для доступа к данным.

Данные в языке Python представлены в форме объектов. То есть объект — это участок памяти с определенным значением и возможными операциями его обработки. Каждый объект имеет свой тип, например int (целое число), str (строка) и др.

В языке Python существуют базовые объектные типы (встроенные в язык) и разрабатываемые пользователем средствами самого языка или другими средствами. Отметим, что переменные хранят не сам объект, а ссылку на объект, то есть адрес объекта в памяти компьютера.

В программе на языке Python связь между данными и переменными устанавливается с помощью оператора присваивания, который обозначается знаком (=).

<имя переменной> = <значение переменной>

Выполняется оператор стандартным образом: сначала вычисляется выражения справа от знака равенства, а затем полученное значение записывается в переменную, указанную слева от знака равенства.

Тема 2.3. Типы данных в Python

Практически любая программа получает на вход какие-либо данные, обрабатывает их и генерирует выходную информацию. Для этого типизированный язык программирования должен знать, с какими данными он имеет дело и какие операции над ними можно производить.

Язык Python имеет множество встроенных типов данных. Все типы делятся на простые и составные.

Переменные простых типов состоят из единственного значения, к ним относятся числа и логические переменные.

Числа в Python делятся на:

- целые,
- действительные (с плавающей точкой размером в 64 бита),
- комплексные (с плавающей точкой размером в 128 бит).

Переменные составных типов состоят из набора значений и, в свою очередь, делятся на неизменяемые, для которых нельзя изменять значения элементов, добавлять или изымать элементы, и изменяемые, для которых всё это можно делать.

К неизменяемым типам относятся строки и кортежи, к изменяемым — списки, словари и множества.

Основными типами данных в Python являются:

- **int** – целочисленные значения – положительные и отрицательные целые числа, а также 0 (например, 4, 687, -45, 0);
- **float** – вещественные (дробные) значения (например, 1.45, -3.789654, 0.00453). Для разделения целой и дробной частей здесь используется точка, а не запятая;
- **bool** – логические значения — значения могут принимать одно из двух значений: True (истина) или False (ложь);
- **str** – символьная строка или единичный символ
- **набор символов**, заключенных в кавычки (например, "ball", "Whatisyourname?", 'd', '6589').

Кавычки в Python могут быть одинарными или двойными.

В языке Python применяется динамическая типизация данных. Это значит, что не нужно объявлять типы переменных, как это делается во многих языках программирования, поскольку их тип определяется автоматически в процессе присваивания им значений. Также, автоматически освобождается память от тех данных, которые становятся ненужными.

Этот тип определяется значением, расположенным справа от оператора присваивания (=).

```
>>>x = 44 # переменная x имеет тип int и значение 44
```

В одной строке можно присвоить одинаковые значения нескольким переменным, например:

```
>>> y = z = 1.10 # переменные y и z имеют тип float и значение 1.10
```

Если необходимо поменять значения местами, чтобы x = 1.10, а y = 44, тогда достаточно сделать так:

```
>x,y = y,x
```

Так же полезно запомнить, что для проверки типа любого значения и любой переменной можно использовать функцию type():

```
>>> a=5
>>> b=17.1
>>> c=4+2j
>>> type(a); type(b); type(c) 4.
```

Тема 2.4. Ввод и вывод данных

Ввод данных осуществляется при помощи оператора input():

```
a = input()
```

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

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

Функция input воспринимает входные данные, как поток символов.

```
a = input('Введите первое число: ')
b = input('Введите второе число: ')
print(a + b)
```

Ход работы программы:

```
Введите первое число: 10
Введите второе число: 4 104
```

Поэтому, чтобы принять целочисленное значение, следует воспользоваться функцией int():

```
a = int(input('Введите первое число: '))
b = int(input('Введите второе число: '))
print(a + b)
```

Вывод программы:

```
Введите первое число: 10
Введите второе число: 4 14
```

Вывод данных осуществляется при помощи оператора print():

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

Внутри круглых скобок через запятую пишется то, что хотим вывести.

По умолчанию функция print() принимает несколько аргументов, выводит их через пробел, после чего ставит перевод строки.

Это поведение можно изменить, используя именованные параметры sep (разделитель) и end (окончание).

В частности, если end сделать пустой строкой, то print не перейдет на новую строчку, и следующий print продолжит вывод прямо на этой же строке.

```
print('При')
print('вет!')
# эти две строки кода выведут "При" и "вет!" на отдельных строках
print('При', end='')
print('вет!') # эти две строки кода выведут "Привет!"
print('Раз', 'два', 'три') # выведет "Раз два три"
print('Раз', 'два', 'три', sep='--') # выведет "Раз--два--три"
```

Для форматированного вывода используется format:

```
x = 11 print ( "{:4d}".format(x) )
```

В результате выведется число 11, а перед ним два пробела, так как указано использовать для вывода четыре знакоместа.

Экранирующая последовательность. Если внутри кавычек встречается символ \ – обратная косая черта, обратный слеш, бэкслеш, он вместе с идущим после него символом образует экранирующую последовательность (escape sequence) и воспринимается интерпретатором как единый специальный символ. В частности, \n — символ начала новой строки, \t — табуляция.

Тема 2.5. Операторы и выражения

Операции над объектами выполняются с помощью соответствующих операторов. Объекты, над которыми выполняются операции, называют операндами.

Оператор в Python — это символ, который выполняет операцию над одним или несколькими операндами. Каждый оператор может выполнять операции над строго определенными для него типами операндов.

В зависимости от типа объектов, над которыми выполняются операции, операторы группируются в арифметические, логические, сравнения, присваивания и др.

Операторы Python бывают 7 типов:

- Арифметические операторы
- Операторы сравнения
- Операторы присваивания
- Логические операторы
- Операторы принадлежности
- Операторы тождественности
- Битовые операторы

Арифметические операторы предназначены для выполнения операций над числами (таблица 1).

Таблица 1 - Арифметические операторы

Оператор	Описание
+	Сложение

-	Вычитание
*	Умножение
/	Деление
**	Возведение в степень
//	Целочисленное деление. 25 // 6 в результате будет 4
%	Остаток от целочисленного деления. 7 % 2 в результате будет 1

Операторы сравнения сравнивают значения объекта, который находится слева от оператора, со значением объекта, который расположен справа от этого оператора. Если условие выполняется, возвращается значение True, иначе — False. Состав и обозначения операторов сравнения приведены в таблице 2.

Таблица 2 - Операторы сравнения

Оператор	Описание
==	Равно
!=	Не равно
>	Больше
<	Меньше
>=	Больше или равно
<=	Меньше или равно

Оператор присваивания (таблица 3) присваивает значение переменной. Он может манипулировать значением до присваивания. Есть простой оператор присваивания и операторы с использованием арифметических операторов.

Таблица 3 - Операторы присваивания

Оператор	Описание
=	присваивание
+=	сложение и присваивание. x+= 8 (эквивалентно x = x + 8)
-=	вычитание и присваивание. x- = 8 (эквивалентно x = x - 8)
*=	умножение и присваивание. x*= 8 (эквивалентно x = x * 8)

/=	деление и присваивание. x/= 8 (эквивалентно x = x / 8)
**=	возведение в степень и присваивание. x**= 8 (эквивалентно x = x ** 8)

Логические операторы (таблица 4) выполняются над данными логического типа, имеют два значения: True (истинное) и False (ложно).

Таблица 4 - Логические операторы

Оператор	Описание
and	Логический оператор "И"
or	Логический оператор "ИЛИ"
not	Логический оператор "НЕ"

Операторы принадлежности (таблица 5) проверяют, является ли значение частью последовательности. Они предназначены для проверки наличия элемента в составных типах данных, таких, как строки, списки, кортежи или словари.

Таблица 5 - Операторы принадлежности

Оператор	Описание
in	Проверяет, является ли значение членом последовательности
not in	проверяет, НЕ является ли значение членом последовательности

Операторы тождественности (таблица 6) проверяют, являются ли операнды одинаковыми (занимают ли они одну и ту же позицию в памяти).

Таблица 6 - Операторы тождественности

Оператор	Описание
is	Возвращает истину, если элемент присутствует в последовательности, иначе возвращает ложь.
is not	Возвращает истину, если элемента нет в последовательности.

Побитовые операторы (таблица 7) предназначены для работы с данными в битовом (двоичном) формате. Эти операторы работают над операндами бит за битом. Предположим, что у нас есть два числа a = 60; и b = 13. В двоичном формате они будут иметь следующий вид:

a = 0011 1100

b = 0000 1101

Таблица 7 – Битовые операторы

Оператор	Описание
&	Бинарный "И" оператор, копирует бит в результат только если бит присутствует в обоих операндах
|	Бинарный "ИЛИ" оператор копирует бит, если тот присутствует в хотя бы в одном операнде
^	Бинарный "Исключительное ИЛИ" оператор копирует бит только если бит присутствует в одном из операндов, но не в обоих сразу
~	Бинарный комплиментарный оператор. Является унарным (то есть ему нужен только один операнд) меняет биты на обратные, там где была единица становиться ноль и наоборот
<<	Побитовый сдвиг влево. Значение левого операнда "сдвигается" влево на количество бит указанных в правом операнде
>>	Побитовый сдвиг вправо. Значение левого операнда "сдвигается" вправо на количество бит указанных в правом операнде

Операторы используются в выражениях. Понятие выражения в программировании аналогичное понятию выражения в математике.

Выражение в языке программирования состоит из операндов, операторов и круглых скобок и определяет порядок выполнения операций над данными.

Операнды выражения — это переменные, константы, функции, методы. Самое простое выражение состоит из одного операнда, например константы или переменной.

В зависимости от типа операндов и операций, используемых в выражении, различают выражения: арифметические (результат арифметического типа), логические (результат логического типа) и строчные (результат строчной типа). Для каждого типа операций существуют четкие правила их записи и исполнения.

В состав Python встроены функции:

abs(x) - Модуль числа x

int(x) – приведение вещественного числа x к целому, отбрасывание дробной части;

round(x) – округление вещественного числа x к ближайшему целому

pow(x, y) - полный аналог записи x ** y

len(x) - Возвращает число элементов в указанном объекте

max() - Максимальный элемент последовательности.

min() - Минимальный элемент последовательности.

sum() - Сумма элементов последовательности.

Помимо стандартных выражений для работы с числами (а в Python их не так уж и много), в составе Python есть несколько полезных модулей. Например, модуль math предоставляет более сложные математические функции, модуль random реализует генератор случайных чисел и функции случайного выбора и др.

Загрузка модулей в Python осуществляется с помощью оператора import. import math # далее используем какую-либо функцию: t = math.sin(math.pi/6) print (math.sqrt(64)) # 8.0

Если название модуля слишком длинное, или оно вам не нравится по каким-то другим причинам, то для него можно создать псевдоним с помощью ключевого слова as:

import matplotlib.pyplot as plt plt.plot(x)

Модуль math – один из наиважнейших в Python. Этот модуль предоставляет обширный функционал для работы с числами. Наиболее употребительные функции и константы модуля math:

trunc(X) - Усечение значения X до целого

ceil() - округление числа до ближайшего наибольшего целого

floor() - округление числа до ближайшего наименьшего целого

sqrt(X) - Квадратный корень из X

exp(X) - Экспонента числа X

log(X), log2(X), log10(X) - Натуральный, двоичный и десятичный логарифмы

X log(X, n) - Логарифм X по основанию n

sin(X), cos(X), tan(X) - Синус, косинус и тангенс X, X указывается в радианах

factorial(X) - Факториал числа X

pi - Выдаётся число pi

e - Выдаётся число e

Более подробное описание этих функций модуля math можно найти на сайте с документацией языка Питон или по ссылке https://pythonworld.ru/moduli/modulmath.html .

Случайные числа. Модуль random

Иногда в программах, например, таких как компьютерные игры или лотереи, требуется получить какое-то случайное число. Для получения случайных чисел в Python используется модуль **random.** Одна из стандартных функций модуля - функция **randint** генерирует случайное целое число. Как мы уже знаем, сначала записывается имя самого модуля, а затем через точку функция с ее аргументами.

```
import random
print (random.randint(1, 20))
>>>
```

Функция **randint()** выбрала случайное число в диапазоне от первого до второго числа в скобках. В нашем примере она выбрала число 7.

Глава 3. Основные алгоритмические инструкции языка Python

Основные структуры алгоритмов (ОСА) – это определенный набор блоков и стандартных способов их соединения для выполнения типичных последовательностей действий. Любой алгоритм может быть записан с помощью трёх алгоритмических конструкций: последовательного выполнения команд (линейных алгоритмов), условных операторов и циклов. Важно знать, что в языке Python синтаксис обладает следующей особенностью: дело в том, что в коде нет операторных скобок (begin..end или {...}); вместо них отступы указывают, какие операторы выполнять внутри той или иной конструкции.

Тема 3.1. Линейный алгоритм (последовательность действий)

Алгоритм называется линейным, если он содержит N шагов, и все шаги выполняются последовательно друг за другом от начала до конца. Для реализации алгоритмов линейной структуры используются следующие операторы:

1) оператор input() - осуществляет ввод данных;
2) оператор print() - осуществляет вывод данных;
3) оператор присваивания (=) - устанавливает связь между данными и переменными.

Последовательные действия описываются последовательными строками программы. Все операторы, входящие в последовательность действий, должны иметь один и тот же отступ.

Например:

```
a = 1
b = 2
a = a + b
b = a - b
a = a - b
print (a, b)
```

Пример: Линейный алгоритм.

Алгоритм вычисления значения выражения K=3b+6a (рис.1) .

Тема 3.2.Разветвляющийся алгоритм

До сих пор мы учились вводить данные и присваивать значения переменным. Теперь надо научиться организовывать различные потоки выполнения приложения в зависимости от ситуации, которая складывается в ходе работы программы.

Алгоритм называется разветвляющимся, если последовательность выполнения шагов алгоритма изменяется в зависимости от выполнения некоторых условий. Условие - это логическое выражение, которое может принимать одно из двух значений: «ДА» - если условие верно (истинно), и «НЕТ» - если условие неверно (ложно).

Разветвляющийся алгоритм можно реализовать в программах с помощью простого, сокращенного, составного операторов, а также конструкции многозначных ветвлений. Рассмотрим их более подробно.

Простой условный оператор

Общий вид в алгоритме конструкции простого условного оператора представлен на рис. 1.

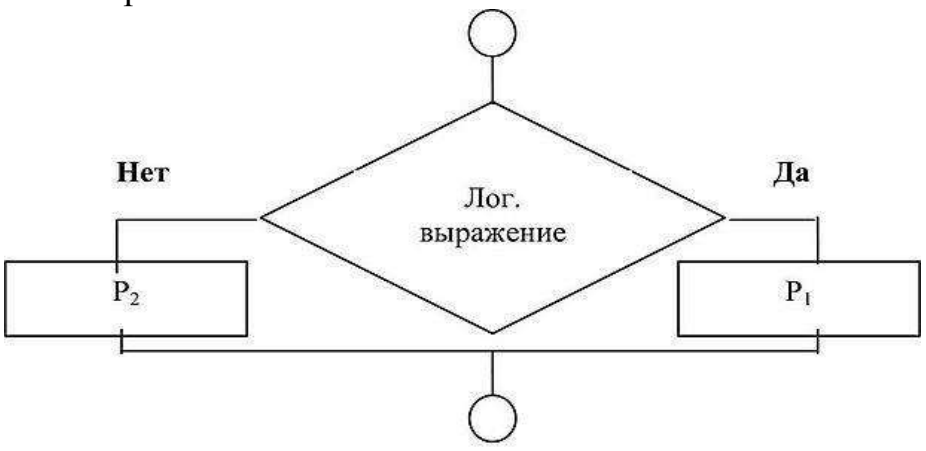


Рис.1.Блок - схема простого условного оператора

Синтаксис простого условного оператора следующий:

if логическое выражение:

P1

else:

P2

где if (если), else (иначе) - зарезервированные слова, а P1, P2 - операторы.
Простой условный оператор работает по следующему алгоритму: сначала вычисляется логическое выражение. Если результат есть true (Истина), то выполняется оператор P1, а оператор P2 пропускается. Если результат есть false (Ложь), то выполняется оператор P2, а оператор P1 пропускается.

Например, в следующем коде в зависимости от результата сравнения двух переменных выводится тот или иной ответ. Оператор if - это блочный оператор, поэтому отступы в коде обязательны. В Python в конце заголовков инструкций с условным оператором всегда ставится двоеточием.
Рассмотрим их работу на примерах.
Задача 1. Напишем программу, разрешающую доступ к сдаче тестов на получение водительских прав только для тех, чей возраст достиг 18 лет.
программа:

```
a = int(input('Введите свой возраст: '))
if a >= 18:
print('Доступ к тесту разрешен')
else:
print('Доступ к тесту запрещен')
```

Как видим, после оператора if записано само условие: a >=18, которое называется заголовком условного оператора. Если данное условие будет верным(True), то выполняется строка кода, идущая сразу за оператором: print('Доступ к тесту разрешен'). Если нет условие будет неверным (False), то эта строка не будет выполнена, а программа сразу перейдет к исполнению команд, стоящих после оператора else. В нашем примере это: print('Доступ к тесту запрещен').

Сокращенный условный оператор

Если необходимо выполнить некоторое действие только при истинности проверяемого условия, то в таком случае применяется сокращенный условный оператор. Общий вид в алгоритме конструкции сокращенного условного оператора представлен на рис. 2.

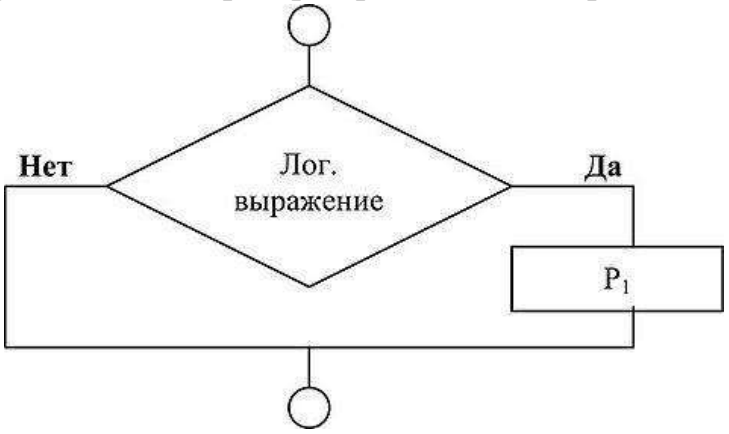


Рис. 2. Блок-схема сокращенного условного оператора

Синтаксис сокращенного условного оператора следующий:
If lor. выражение:
P1, где if (если) - зарезервированное слово, а P1- оператор.

Например, при истинности логического выражения выводится соответствующее сообщение «Доступ к тесту разрешен».
программа:
a = int(input("Введите значение a "))
if a >= 18:
print('Доступ к тесту разрешен')

Составной условный оператор

Если при некотором условии надо выполнить определенную последовательность операторов, то их объединяют в один составной оператор. Общий вид в алгоритме конструкции составного условного оператора представлен на рис. 3.

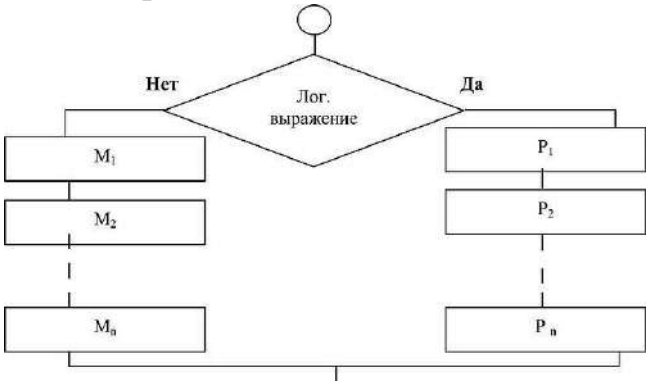


Рис. 3. Блок-схема составного условного оператора

Синтаксис составного условного оператора следующий:

if - ог. выражение:
P_1,
P_2
.
P_n
else:
M_1,
M_2
.
M_n

где if, else - зарезервированные слова, а P_1, P_2,... P_n, M_1,M_2... M_n- операторы.

Например, в каждой ветви условного оператора мы хотим выполнить по два оператора. Тогда каждый из них мы должны сместить вправо ровно на четыре пробела от начала строки.

a = int(input("Введите значение a"))
b = int(input("Введите значение b"))
if a < b:
a=a+b
print("Сумма двух чисел =", a)

```
else:
a=a*b
print("Произведение двух чисел=", a)
```

Тема 3.3. Многозначные ветвления

Очень часто приходиться выбирать путь решения задачи не из двух, а из нескольких возможных. В программировании данный ход можно реализовать, используя несколько условных операторов, дополнительный блок elif(сокращенное от else - if). Общий вид в алгоритме конструкции многозначных ветвлений представлен на рис. 4.

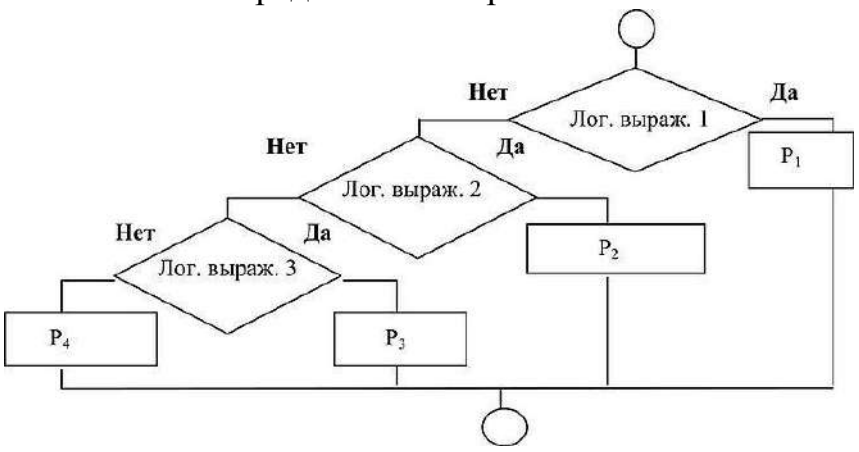


Рис. 4. Блок-схема конструкции многозначных ветвлений

Синтаксис условного оператора в конструкции многозначных ветвлений следующий:

```
if лог. выражение 1:
P1.
elif лог. выражение2:
P2
elif лог. выражение3:
P3
else
p4
```

где if, elif, else - зарезервированные слова, а Pi, P2, P3, P4 - операторы.

Алгоритм работы такой конструкции состоит в следующем. Если логическое выражение истинно, то выполняется оператор или блок операторов, следующих в данной ветви, в противном случае этот оператор или блок пропускается. Если логическое выражение, следующее за оператором if, ложно, то анализируется Логическое выражение2, следующее за оператором elif. Если оно истинно, то выполняется оператор или блок операторов, следующих в данной ветви, в противном случае этот оператор или блок пропускается. Операторы, следующие за последним else, выполняются лишь в том случае, если ложны все предыдущие логические выражения. Условные операторы if в такой конструкции называются вложенными.

Например, в программе показан процесс тестирования трех ветвей программы. Пользователь может менять исходные данные, которые будут находиться в ячейках a и b, и всякий раз получать тот или иной результат.
программа:

```
a = int(input("Введите значение a "))
b = int(input("Введите значение b "))
if a < b: a=a+b
print("Сумма двух чисел =", a)
elif a=b: a=a*b
print("Произведение двух чисел =", a) else: a=a-b
print("Разность двух чисел =", a)
```

Алгоритмы поиска максимального и минимального элементов

Рассмотрим алгоритмы нахождения максимального и минимального значений, которые будут востребованы при дальнейшем изучении языка программирования Python.

Задача 1. **Найдите максимальное из двух чисел**. Разработка алгоритма решения задачи представлена на рис. 5.

Программа:

```
a = int(input("Введите значение a "))
b = int(input("Введите значение b "))
if a >= b:
max=a
else:
max=b
print("Максимальное из двух чисел =", max)
```

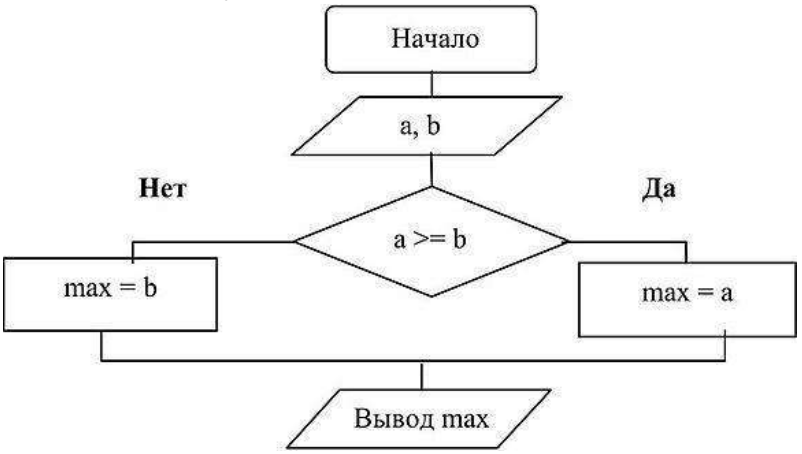


Рис. 5. Алгоритм нахождения максимального из двух чисел

Задача 2. **Найдите минимальное из трех чисел**. Алгоритм нахождения максимального или минимального элемента может быть запрограммирован несколькими способами. Фрагмент разработки алгоритма решения задачи представлен на рис. 6. Очевидно, что при большом количестве чисел, из которых нужно осуществить выбор экстремального значения, алгоритм станет громоздким и запутанным .

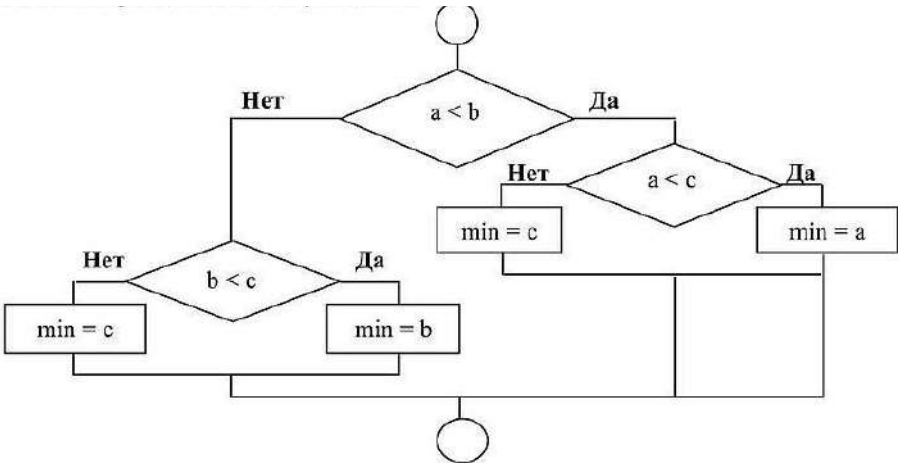


Рис.6. Алгоритм нахождения минимального из трех чисел .

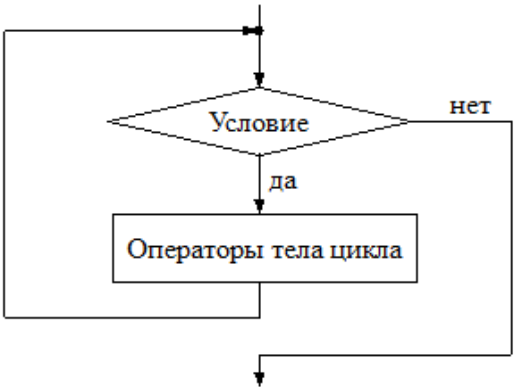
Тема 3.4. ЦИКЛИЧЕСКИЙ АЛГОРИТМ

Алгоритм называется циклическим, если определенная последовательность шагов выполняется несколько раз в зависимости от заданной величины, которая называется параметром цикла. Цикл заканчивается, когда параметр принимает определенное значение.

Циклы являются такой же важной частью программирования, как условные операторы. С их помощью можно организовать повторение некоторых частей кода. В Python для записи циклов используются 2 вида команд: **while и for**.

Цикл while

While переводится с английского как «пока», то есть цикл (блок команд) выполняется до тех пор, пока не выполнится заданное условие. Для этого в начале очередного шага цикла выполняется проверка условия. Поэтому оно называется циклом с предусловием. Как он выглядит в виде блок-схем:



Ромбом выделен заголовок цикла с условием, прямоугольником - тело цикла, которое состоит из серии команд. Если при старте цикла, условие в заголовке верно(TRUE), то команды в теле цикла исполняются один раз, и программа снова возвращается в основную ветку. Здесь условие снова проверяется, и если оно верно, то блок команд выполняется второй раз. Потом снова возвращается к заголовку и так далее. Так процесс повторяется до тех пор, пока условие цикла будет возвращать “истину”.

Цикл завершает свою работу только тогда, когда логическое выражение в заголовке возвращает “ложь”, то есть условие выполнения цикла больше не соблюдается. После этого поток выполнения перемещается к выражениям, расположенным ниже всего цикла. Говорят, “происходит выход их цикла”.

Задача 1. Запишем программу для вывода на экран всех целых чисел от 1 до 5.

Программа на Rython:

```
d=1
while d<=5:
    print (d)
    d=d+1
```

Результат:

```
1
2
3
4
5
```

Как и в случае с оператором if, условие цикла записывается сразу после слова while. В нашем примере это: d<=5. Обязательно в конце условия нужно поставить двоеточие “:”, а команды в теле цикла записать с отступом вправо.

В рассматриваемой программе использована переменная-счетчик d. Ее начальное значение равно 1. С каждым проходом цикла ее значение будет увеличиваться на 1. Для этого мы записали строку d=d+1.Цикл остановится тогда, когда значение переменной достигнет пяти.

Задание 2. Напишем программу,которая просит пользователя ввести двухзначный код и затем сравнивает его с верным кодом. Если пользователь вводит неверный код, то программа запрашивает новый. Программа с использованием цикла while будет выглядеть так:

Программа на Rython:

```
code = 35
a = int(input('Введите двузначный код:'))
while a != code:
    print ('Неверный код.Введите снова.')
    a = int(input('Введите двузначный код:'))
    if a == code:
        print ('Код введен верно')
```

Результат:

```
Введите двузначный код:45
Неверный код.Введите снова.
Введите двузначный код:95
Неверный код.Введите снова.
Введите двузначный код:75
Неверный код.Введите снова.
Введите двузначный код:35
Код введен верно
```

Зашифрованный программой код равен 35. Как видно из условия (while a != code:), пока пользователь не введет число равное 35, программа будет просить ввести код заново. В программе появилось условие if, которое указывает, что если введенное число будет равно зашифрованному коду(if a == code:) , то программа завершится.

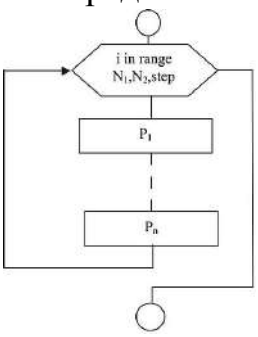
Самое главное для цикла while - чтобы хоть когда нибудь наступил случай, когда логическое выражение в заголовке возвращает FALSE. Иначе цикл будет работать бесконечно долго, и произойдет зацикливание.

Цикл for

Очень часто в программах надо выполнить определенные операторы несколько раз. Нелогично записывать эту последовательность действий двадцать или пятьдесят раз подряд. В этих случаях организуют

циклические вычисления. Для организации циклов с известным количеством повторений в языке Python используется оператор for.

Его общий вид в алгоритме представлен на рис. 1



Цикл for повторяет команды необходимое количество раз. Данная команда позволяет сделать программу компактнее. Рассмотрим предыдущий пример с использованием цикла for:

Программа на Rython: **Результат:**

```
for i in range (5):
    print(i)
```

```
0
1
2
3
4
```

Здесь переменная i (ее называют переменной цикла) изменяется в диапазоне (in range) от 0 до 5, не включая 5 (то есть от 0 до 4-х включительно). Таким образом, цикл выполняется ровно 5 раз.

Для того чтобы нам получить идентичный ответ (как в случае с while), изменим нашу программу:

Программа на Rython: **Результат:**

```
for i in range (1,6):
    print(i)
```

```
1
2
3
4
5
```

Аргументы для функции **range()** передаются следующим образом:

range(x) – перебирает все значения от 0 до x; при этом число x не включается в диапазон;

range(y, x) – перебирает все значения от y до x; x также не включается в диапазон;

range(y, x, s) – перебирает значения от y до x, с шагом s.

Задача 1. Выведем степени числа два от 2^1 до 2^{10} (k = степени двойки).

Программа на Rython: **Результат:**

Равносильные записи выражения

```
k = 1
while k <= 10:
    print (2**k)
    k+=1
```

```
for k in range (1,11):
    print(2**k)
```

```
2
4
8
16
32
64
128
256
512
1024
```

В первом варианте переменная k используется трижды: для присвоения начального значения, в условии цикла и в теле цикла (увеличение на 1). Во втором варианте переменная k задается диапазоном (range) из двух чисел – начальным и конечным значением, причем конечное значение не входит в диапазон.

Шаг изменения переменной цикла по умолчанию равен 1. Если его нужно изменить, указывают третье (необязательное) число в скобках после слова range – это нужный шаг. Например, такой цикл выведет только нечетные степени числа 2:

Программа на Rython:

```
for k in range (1,11,2):
    print(2**k)
```

Результат:

```
2
8
32
128
512
```

С каждым шагом цикла переменная цикла может не только увеличиваться, но и уменьшаться. Для этого начальное значение должно быть больше конечного, а шаг – отрицательным. Следующая программа печатает квадраты натуральных чисел от 5 до 1 в порядке убывания:

Программа на Rython:

```
for k in range (5,0,-1):
    print(k**2)
```

Результат:

```
25
16
9
4
1
```

Рассмотрим пример, в котором переменной letter каждый раз присваивается новый элемент из строки Python. Команда print выводит на экран каждую букву этой строки по одной:

Программа на Rython:

```
for letter in 'python':
    print ('Буква:',letter)
```

Результат:

```
Буква: p
Буква: y
Буква: t
Буква: h
Буква: o
```

Вложенные циклы

В сложных задачах часто бывает так, что на каждом шаге цикла нужно выполнять обработку данных, которая также представляет собой циклический алгоритм. В этом случае получается конструкция «цикл в цикле» или «вложенный цикл».

Цикл называется **вложенным,** если он размещается внутри другого цикла. На первом проходе внешний цикл вызывает внутренний, который исполняется до своего завершения, после чего управление передается в

тело внешнего цикла. На втором проходе внешний цикл опять вызывает внутренний. И так до тех пор, пока не завершится внешний цикл.

Задача 1. Выведем на экран таблицу умножения. Для этого во внешнем цикле надо перебрать числа от 1 до 9. Для каждого из них нужно перебрать во внутреннем цикле числа от 1 до 9.

1 2 3 4 5 6 7 8 9
2 4 6 8 10 12 14 16 18
На одну синюю цифру приходится один ряд черных цифр до 9-ти.
Синие 1 и 2 находятся во внешнем цикле, черные цифры во внутреннем.

При этом во внутреннем цикле нужно выполнить умножение переменных счетчиков внешнего и внутреннего цикла первого ряда. Таким образом, на одно выполнение внешнего цикла произойдет девять выполнений внутреннего, и сформируется одна строка таблицы умножения. После каждой строки надо перейти на новую: это делается во внешнем цикле, после того как закончится выполняться внутренний.

Также для построения таблицы необходимо использовать форматированный вывод, т.е. задавать ширину столбцов (\t), иначе произойдет сдвиг, т.к. количество цифр в каждой строке различно. Наш код будет выглядеть так:

Программа на Rython:

```
for i in range(1,10):
    for j in range(1,10):
        print(i*j, end='\t')
    print()
```

Результат:

1	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18
3	6	9	12	15	18	21	24	27
4	8	12	16	20	24	28	32	36
5	10	15	20	25	30	35	40	45
6	12	18	24	30	36	42	48	54
7	14	21	28	35	42	49	56	63
8	16	24	32	40	48	56	64	72
9	18	27	36	45	54	63	72	81

Задача 2. Нарисуем на экране «прямоугольник» из двух видов символов. Края прямоугольника будут отрисованы символом «0», а внутренняя часть символом «1». Пусть длина прямоугольника будет равна 10 символам, а ширина 7. Внешний цикл, перебирая строки, первой и последней цифрой должен поставить 0. Если строка первая или последняя (поскольку отсчет идет от 0, то это 0-я и 6-я строка), 0 выстраивается от начала и до конца. Во всех остальных случаях – ставим цифру 1. Запишем программу:

Программа на Rython:

```
for i in range(7):
    if i==0 or i==6:
        for j in range (20):
            print ('0',end='')
    else:
        print ('0',end='')
        for j in range(1,19):
            print('1',end='')
        print('0',end='')
    print()
```

Результат

```
00000000000000000000
01111111111111111110
01111111111111111110
01111111111111111110
01111111111111111110
01111111111111111110
00000000000000000000
```

Операторы break и continue

Иногда нам может понадобиться, чтобы при определенных условиях программа принудительно вышла из цикла. Для таких случаев используется оператор **break.** Если же будет необходимо, чтобы цикл пропустил только какой-либо определенный проход и дальше

продолжался, то используется оператор **continue.** Сравним работу операторов на примере:

Программа на Rython:

```
for i in [1,2,3,4,5]:
    if i == 3:
        break
    print('элемент списка',i)
```

Результат:

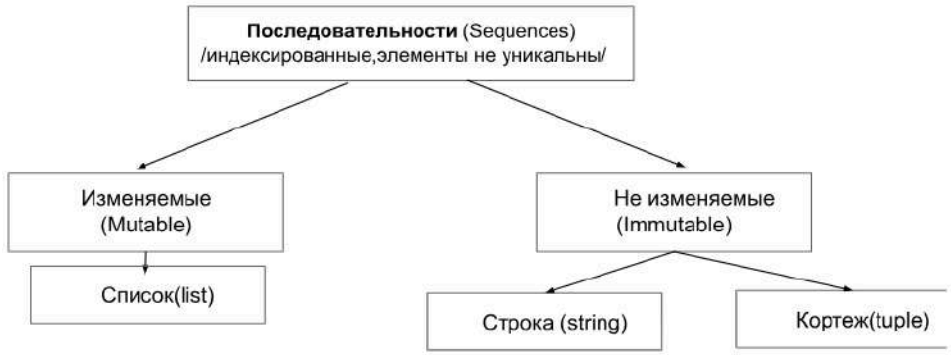
```
элемент списка 1
элемент списка 2
```

```
for i in [1,2,3,4,5]:
    if i == 3:
        continue
    print('элемент списка',i)
```

```
элемент списка 1
элемент списка 2
элемент списка 4
элемент списка 5
```

Глава 4. Коллекции в Python

Коллекция в Python — программный объект (переменная-контейнер), хранящая набор значений одного или различных типов, позволяющий обращаться к этим значениям, а также применять специальные функции и методы, зависящие от типа коллекции. Коллекции в Python делятся на последовательности, множества и отображения (рис.1). Среди встроенных типов данных к первым относятся списки (тип данных list) и кортежи (tuple), ко вторым – обычные (изменяемые) и фиксированные множества (set и frozenset), к третьим – словари (dict).



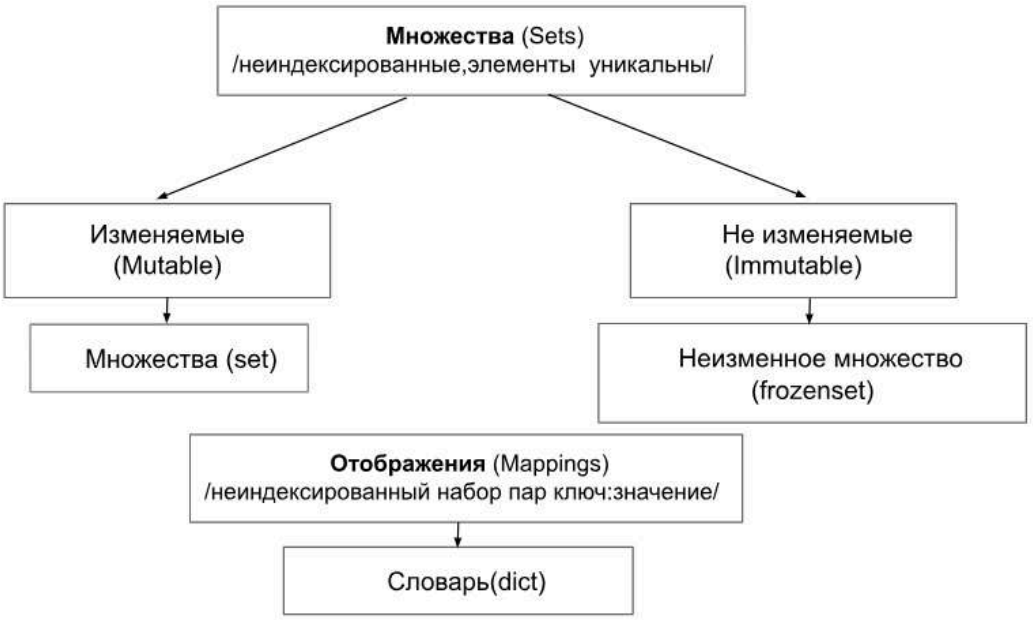


Рисунок 1 – Классификация коллекций в Python

Узнать тип любого объекта можно с помощью встроенной функции **type()**. Основными свойствами для характеристики коллекций являются:

- **Индексированность** – каждый элемент коллекции имеет свой порядковый номер — индекс. Это позволяет обращаться к элементу по его порядковому индексу, проводить слайсинг («нарезку») — брать часть коллекции, выбирая исходя из их индекса.

- **Уникальность** – каждый элемент коллекции может встречаться в ней только один раз. Это порождает требование неизменности используемых типов данных для каждого элемента, например, таким элементом не может быть список.

- **Изменяемость коллекции** — позволяет добавлять в коллекцию новых членов или удалять их после создания коллекции.

Тема 4.1.Строки и операции с ними.Методы строк.
Тема 4.2.Сравнение и сортировка строк. Форматирование строк
Тема 4.3. Множества в Python
Тема 4.4.Списки в Python
Тема 4.5.Кортежи в Python
Тема 4.6.Словари в Python

Методическая разработка аудиторных форм работы (Краткое содержание практических занятий)

1.Практическая работа №1. Введение в язык программирования Python. Изучить основные типы данных, команды ввода и вывода данных. -
https://docs.google.com/document/d/1pDYqL49PgphHRQ9a_rjzIQJFjGlPpMrQ7-pT7TaE3I4/edit?usp=sharing

2.Практическая работа № 2. Линейный алгоритм: Математические операции в Python.Библиотека (модуль) math -

https://docs.google.com/document/d/1zx2bxvraL7KqsgbuSGUK00FF9Nj8JGWM3I5PFsbuDu0/edit?usp=sharing

3.Практическая работа № 3. Разветвляющиеся алгоритм: структура ветвлене в Python. Множественное ветвление. - https://docs.google.com/document/d/1cnAYVziI_03qM5nur6aW5jd5SI9a6etjTEx4zrqt2Go/edit?usp=sharing

4.Практическая работа №4. Циклический алгоритм: Работа с циклами в Python: цикл while, цикл for. https://docs.google.com/document/d/1ryJm3himeeIvw1Han8s2T-bqBcrpxK2yL3FRquDBQZE/edit?usp=sharing

Критерии баллов — рейтинговой оценки знаний и умений студентов.

Деятельность студентов в течение семестра оценивается следующим образом: работа на семинарах (50%), самостоятельные работы и реферат (20%), активность (25%), посещение занятий (5%).

Работа на семинарах (50%)

Чтение текстов и участие в дискуссиях являются важными составляющими работы на семинарах. Приветствуются вопросы по структуре и содержанию текста, комментарии, помогающие уяснить значение основных категорий и т.п.

Пропущенные семинары необходимо отработать письменно. «Отработка» должна содержать основные моменты пропущенной темы занятия. Оценка за «отработки» не выставляется. Последний срок сдачи «отработок» - заключительное занятие по курсу (тем, кто не сможет присутствовать на заключительном занятии «отработку» необходимо принести заранее).

Неотработанные семинары являются основанием незачета по данному курсу.

Критерии оценки: регулярное присутствие и активное участие, уместность и глубина вопросов и комментариев, способность задавать живой импульс дискуссии и вовлекать других студентов в дебаты.

Оценки за активность на семинарах выставляются по 10-ти балльной шкале.

Критерии оценки работы студентов на семинарах следующие:

10 баллов – индивидуальный ответ, изложенный по существу структурно, логично, своими словами.

8-9 баллов – индивидуальный ответ, изложенный своими словами. Возможны мелкие проблемы с логикой изложения.

5-7 баллов – индивидуальный ответ, изложенный частично своими словами. Возможны мелкие проблемы с логикой изложения.

1-4 балла – индивидуальный ответ – уточнение (дополнение) по рассматриваемым вопросам семинарского занятия, задаваемые вопросы.

Самостоятельные работы и реферат (20%)

Самостоятельные работы выполняются на отдельном листочке письменно от руки. Указывается имя, фамилия, группа и дата сдачи работы.

Все письменные работы НЕ принимаются позже установленных сроков сдачи, за исключением документально подтвержденных случаев отсутствия вследствие болезни или форс-мажорных обстоятельств.

Критерии оценки письменных работ следующие:

10 – выдающаяся работа на высоком уровне, присутствует логика и оригинальность изложения, выдвинут и доказан тезис, видно уверенное владение освоенным материалом.

8-9 – очень хорошая работа, продемонстрированы не только усвоенные знания по курсу, но навыки анализа материала и самостоятельного мышления. Возможны мелкие проблемы с логикой изложения.

6-7 – хорошая работа, продемонстрированы не только усвоение фактических знаний по курсу и основные навыки аргументации, но изложение не вполне закончено с точки зрения обоснования тезиса и раскрытия вопроса.

4-5 – средняя работа, неполное усвоение фактических знаний по курсу, слабая логика изложения и обоснования.

2-3 – плохая работа, отрывочные знания по курсу, слабая логика изложения и обоснования.

1 – отсутствие каких-либо знаний.

0 – доказанный случай плагиата.

Темы рефератов студенты выбирают согласно нумерации по учебному журналу.

Реферативная работа оформляется письменно от руки. Допускается

печатное исполнение титульного листа, списка литературы, графических и табличных приложений.

Студенты, вовремя не сдавшие реферат, защищают свою работу на консультации или в дополнительно отведенное время.

Своевременное выполнение работ является предпосылкой к обоснованию возможности допуска студента к зачету (экзамену).

Проверка уровня усвоения лекционных занятий, включая теоретических СРС и СРСП, проводится тестированием по рейтинго-модульной системе. Каждый тест включает 15 вопросов, где правильный ответ на 1 вопрос оценивается на 1 балл.

Результаты практических работ, включая, практических СРС и СРСП принимаются в виде графических и контрольных работ, рефератов и собеседования.

Формы текущего и итогового контроля

№	Этапы проверки	Вид средства проверки	Балл
1	1 модуль	Проверка практических заданий. Устно, тестирование.Посещаемость.	100
2	2 модуль	Проверка практических заданий. Устно, тестирование. Посещаемость.	100
3	Итоговый контроль: ● Практическое занятие. ● СРС	Тесты и графические работы. Рефераты.Презентации. Самостоятельная работа. Практические задания.	100
	Общий средний балл		**100**

Распределение баллов по модулям

		Удовлетворительно	Хорошо	Отлично
1 модуль – 100 б.		60-79	80-89	90-100
2 модуль– 100 б.		60-79	80-89	90-100
Практические занятие – 50 б.	Итоговый контроль	60-79	80-89	90-100
СРС – 50 б.				

Вопросы к модулям

Контрольные вопросы

1. Этапы решения задач на ЭВМ.
2. Алгоритм. Свойства алгоритма. Формы записи алгоритма.
3. Данные и их типы. Методы сортировки данных.
4. Основные структуры алгоритмов.
5. Логические высказывания и операции.
6. Таблицы истинности. Свойства логических операций.
7. Язык программирования. Поколения, классификация, элементы языков программирования.
8. Система программирования. Классификация СП.
9. Структура системы программирования. Методы программирования.
10. Виды программного обеспечения. Общие принципы разработки ПО. Жизненный цикл ПО.
11. Язык программирования Python. Среда программирования Wing IDE.
12. Основные элементы языка. Типы данных в Python. Операторы и выражения.
13. Для каких целей используются комментарии в программах? Как можно закомментировать участок программного кода в Python?
14. Каково назначение операторов print и input? Приведите примеры использования таких операторов.
15. Программирование алгоритмов линейной структуры.
16. Программирование алгоритмов ветвления. Вложенные условия.
17. Программирование алгоритмов циклической структуры. True и False, break и continue. Вложенные циклы.
18. Расскажите о работе оператора цикла while. Приведите примеры. Расскажите о работе цикла с оператором for по убывающим значениям параметра цикла.
19. Коллекции в Python, их классификация.
20. Строки. Функции и методы строк.
21. Множества. Операции над множествами.
22. Списки. Методы списков. Вложенные списки.
23. Напишите синтаксис объявления списков.
24. Кортежи. Преобразование коллекций.
25. Напишите синтаксис объявления кортежей.
26. Словари. Методы словаря.
27. Напишите синтаксис создания словаря.

Темы для самостоятельной работы студентов

СРС:

1. Разработка программного обеспечения сегодня
2. Алгоритм: способы представления и свойства
3. Языки программирования
4. Классификация языков программирования
5. Выбор языка программирования
6. Почему Python стоит изучать?
7. Алфавит языка Python
8. Основные типы данные в Python.
9. Методы ввода и вывода данных
10. Линейные алгоритмы. Операции над целочисленными данными
11. Запись математических функций, модуль MATH
12. Разветвляющиеся алгоритм
13. Основные алгоритмические инструкции языка Python
14. Многозначные ветвления
15. Циклический алгоритм, оператор цикла FOR
16. Циклический алгоритм, оператор цикла While
17. Вложенные условные операции и циклы
18. Кортежи в Python
19. Методы работы со строками
20. Базовые алгоритмы обработки строк
21. Словари в Python
22. Множества в Python
23. Списки в Python
24. Коллекции в Python, их классификация.

Тестовые задания

Тест за 1 модуль:

1.Алгоритм, записанный на понятном компьютеру языке, называется:

a. исполнителем; b. программой;
c. блок-схемой; d. системой команд исполнителя
e. псевдокодом.

2.Где записана команда присваивания?:

a. X+Y:=X b. F=G
c. X:=X+Y d. A>D

3.Выберите тип величины, который следует использовать для обозначения количества учеников в классе:

a. числовой целый; b. числовой вещественный;
c. строковый; d. логический.

4.Какое арифметическое выражение записано правильно?

a. A1+B1*50; b. 6A-23B;

c. b2 – 4ac d. 67*A2-30*B.

5.Укажите логические выражения:

a.X+7; b. X+7>=0; c. X:=7; d. N=10.

6.Выберите верные утверждения:

a.одна величина может иметь несколько типов;

b. значение переменной может изменяться в процессе выполнения алгоритма;

c. величина логического типа может принимать всего два значения;

d. при присваивании переменной какого-либо значения предыдущее её значение сохраняется автоматически.

7.Как выглядит комментарии в Python:

a. "это пустая программа" b. #это пустая программа

c. "это пустая программа'

8.Что такое "переменная":

a. это величина, имеющая имя, тип и значение. Значение переменной нельзя изменять во время работы программы.

b.это знак проверки условия

c.это величина, значение которой нельзя менять во время работы программы.

d.это величина, имеющая имя, тип и значение. Значение переменной можно изменять во время работы программы.

9.Какие типы переменных используются в Python:

a. Строчные d. Кинетическое число

b. Символьная строка e .Вещественное число

c.Целое число f.Логическая

10.Команда для присваивания нового значения переменной:

a.Оператор Конъюнкция b.Оператор присваивания

c.Условный оператор

11.Что будет в результате работы программы:

```
a=input()
b=input()
c=a+b
print (c)
```

Входные данные a=21 b=33.

a. 54 b. 21 c. 33 d. 2133

12.Как ввести целое число с клавиатуры:

a. a=input() b. a=int(input())

c. a=10 d. a=str(input())

13.Что будет в результате выполнения программы (чему равно a и b):
a=5
b=a+2
a=(a+2)*(b-3)
b=b+1.

a. a=2 b=20 b. a=8 b=28
c. a=28 b=8

14.Возведение в степень в Python:

a. // b. **
c. % d. + e. e^

15.Целочисленное деление:

a. // b. % c. / d. *

16.Что будет в результате такого деления:
a=1234
d=a%10
print(d).

a. 5 b. 4 c. 3 d. 2

17.Что выведет вторая d:
a = 1234
d = a % 10; print(d)
a = a // 10
d = a % 10; print(d).

a. 3 b. 4 c. 5 d. 6

18.Как импортировать библиотеку случайных чисел:

a. from random import b. random.randint
c. random.random d. import random

19.Знак равенства в Python, варианты ответов:

a. != b. >< c. == d.<=

20.Как получить данные от пользователя?

a. Использовать метод read() b. Использовать метод get()
c. Использовать метод cin() d. Использовать метод readLine()
e. Использовать метод input()

21.Какая функция выводит что-либо в консоль (на экран монитора)?

a. out(); b. log();
c. print(); d. write();

22.Какие существуют типы переменных у чисел(выбрать несколько вариантов):

a. float b. list
c. num d. int e. integer

23.Переменная int:

a. вещественная переменная b. символьная строка

c. логическая переменная d. целая переменная

25.Переменная float:

a. целая переменная b. вещественная переменная

c. логическая переменная

26.Имена переменных не могут включать:

a. Русские буквы b. Латинские буквы

c. Пробелы d. Скобки, знаки + = ! ? b др.

27.Какие имена являются правильными в PYTHON (выбрать несколько)^

a. sum b. 41 And A+B

28.Что делает функция math.ceil(X) ?

a. округляет число в большую сторону b.такой функции не существует

c. округляет число в меньшую сторону d. показывает факториал числа X

29.С помощью какой функции можно получить логарифм X по основанию 10?

a. math.log10(X) b. math.log2(10)

c. math.log(10) d. math.logX(10)

30.Что будет выведено на экран:

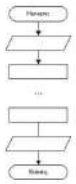
```
import math
print(math.pi)
```

a. 2,718281 b. 3,14

c. 2,74 d. 3,141592

Тест за 2 модуль:

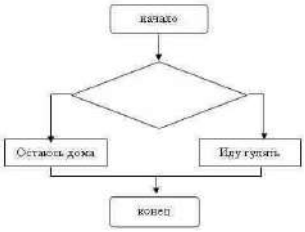
1.Назовите тип алгоритма:



a. Разветвляющийся c. Циклический

b. Линейный d. Смешанный

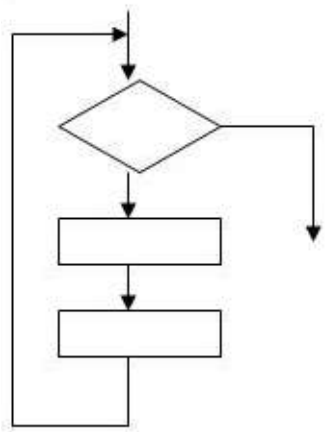
2.Назовите тип алгоритма:



a. Линейный c. Циклический

b. Разветвляющийся d. Смешанный

3.Назовите тип алгоритма:



a. Разветвляющийся
b. Линейный
c. Смешанный
d. Циклический

4.Что хранит в себе переменная?

a. Имя
b. Значение
c. Тип
d. Длину своего значения

5.Какой оператор здесь используется?

```
If n < 100:
b = n + a
```

a. Условный оператор
b. Оператор присваивания
c. Оператор сложения
d. Оператор умножения

6.Что лучше использовать для множественного ветвления?

a. if – elif –else
b. Много if
c. if – else – elif
d. while

7.Оператор цикла в языке Python:

a. while
b. for
c. if
d. print

8.Для чего нужен оператор break?

a. Для завершения программы
b. Для выхода из цикла
c. Для поломки компьютера
d. Для удаления программы

9.Сколько раз программа напишет слово «Пока»?

```
k=0
while k<10:
  print(«Привет»)
  k += 1
```

a. 9
b. 0
c. 10
d. Бесконечно

10.Что такое «else»?

a. Так как
b. Иначе
c. Если
d. Потому что

11.Выберите циклический алгоритм

a.
```
k = 0
while k < 10:
print(“Привет”)
  k += 1
```

b.
```
a = int(input())
b = int(input())
c = int(input())
s = a+b+c
print(c)
```

c.
```
a = int(input())
if a > 0:
  print(a)
else:
  print(a)
```

12.Создатель языка программирования Python

a. Гвидо Ван Россум
b. Дэвид Паттерсон
c. Эрвин Дональд Кнут
d. Джеймс Артур Гослинг

13.На каких операционных системах может работать Python?

a. Windows
b. Linux
c. macOS
d. Ничего из этого

14.Что делает функция len()?

a. Возвращает длину строки
b. Возвращает случайное число
c. Возвращает номер символа
d. Возвращает модуль числа

15.Где находятся параметры, а где аргументы функции?

a. Параметры пишутся при объявлении функции, аргументы при вызове
b. Аргументы пишутся при объявлении функции, параметры при вызове
c. Это одно и то же!
d. У функции есть только параметры

16.Ниже представлены утверждения о списках. Какие из них верны?

a. Один и тот же объект может появляться в списке несколько раз
b. Размеры списка четко не определены
c. Эти два списка одинаковы:
['a', 'b', 'c']
['c', 'a', 'b']
d. Все элементы в списке должны быть одного типа

17.Допустим, объявлен список — a = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']. Ниже представлены несколько программ. В каких из них вывод указан правильно?

a. print(a[4::-2])
['quux', 'baz', 'foo']

b. max(a[2:4] + ['grault'])
'qux'

c.a[:] is a

d.print(a[-5:-3])
['bar', 'baz']

18.Объявлен список — a = [1, 2, 3, 4, 5].

Ниже представлены строки кода, удаляющие элемент. Какие из них в результате дадут список [1, 2, 4, 5]?

a. del a[2]
b. a[2:2] = []
c. a[2:3] = []
d. a.remove(3)
e. a[2] = []

19.Что такое кортеж?

a. Неупорядоченная неизменяемая коллекция объектов произвольных типов
b. Упорядоченная изменяемая коллекция объектов произвольных типов
c. Упорядоченная неизменяемая коллекция объектов произвольных типов
d. Упорядоченная неизменяемая коллекция объектов одного типа

20.Что из следующего ниже верно о кортежах?

a. Кортежи можно складывать
b. Кортеж занимает больше памяти чем список
c. Кортеж может быть ключом словаря
d.У кортежа есть метод append()
e. Кортеж занимает меньше памяти чем список

21.Выберите верные утверждения:

a. Словари изменяемы
b. Словари могут быть любой «глубины»
c. Словарь может содержать объект любого типа, кроме другого словаря
d. Доступ к элементам словаря осуществляется с помощью ключа

22.Ниже представлены несколько вариантов кода. Какой из них удалит элемент с ключом ‘baz’ из словаря?

a. del d['baz']
b. d[‘baz’]
c. del.d(baz)
d. del d(baz)

23.Начинаем с простого. Что такое множество в Python?

a. Это любая коллекция элементов
b. Это список, содержащий в себе только функции
c. Это контейнер, значения в котором не повторяются
d. Это список, содержащий вложенные списки в себе

24.Каким образом правильно объявляется множество?

a. a = {}
c. a = set()
b. a = []
d. a = set

25. Чем отличаются методы remove() и discard(), применяемые к множеству?

a. Ничем
b. remove() удаляет элемент если он есть, но бросает ошибку если элемента нет. discard() просто удаляет элемент если он есть
c. discard() удаляет элемент если он есть, но бросает ошибку если элемента

нет. remove() просто удаляет элемент если он есть
d. Метода discard() для множеств не существует

Список литературы и учебно-методическая литература по дисциплине, разработанная преподавателями отделения

Основные источники:

1. Голицына О.Л., Попов И.И. Основы алгоритмизации и программирования: Уч.пособие.- М.: ФОРУМ: ИНФРА-М, 2004.
2.Златопольский Д. М. Основы программирования на языке Python,Издательство "ДМК Пресс",2018- 396с.
3. Доусон М. Программируем на Python. - СПб.: Питер, 2016. - 416 с.
4. Прохоренок *Н.А*. Python 3 и PyQt. Разработка приложений. - СПб.: БХВ-Петербург, 2015. - 704 с.
5.Цыбуля И.Н. , Самыкбаева Л.Ф. Информатика. - Бишкек, 2020.-203с.
6. Семакин И.Г., Шестаков А.П. Основы программирования: Учебник.- М.: Мастерство; НМЦ СПО; Высшая школа, 2001.
7.Сысоева М. В., Сысоев И. В. Программирование для «нормальных» с нуля на языке Python: Учебник. В двух частях. Часть 1 / Ответственный редактор: В. Л. Черный : — М.: Базальт СПО; МАКС Пресс, 2018. — 176 с. [+4 с. вкл]: ил. — (Библиотека ALT).
8.Федоров Д.Ю. Программирование на языке высокого уровня Python, Учебное пособие для СПО. - Москва:Издательство Юарт , 2019 -211с.

Дополнительные источники:

1. Рапаков Г.Г., Ржеуцкая С.Ю. – Программирование на Python 3/0 для студентов и школьников. – СПб.:БХВ-Петербург, 2007.-352 с.:
2. Ускова О.Ф. – Программирование на языке Питон Задачник. Изд. Питер. 2002. - 336с.
3. Ушаков Д.М., Юркова Т.А. – Python для школьников. –СПб.: Питер, 2006г. – 256с.: 12
4. Чернов А.Ф. – Олимпиадные задачи с решениями и подробным анализом. – Волгоград: Учитель, 2007. – 207с.:

Интернет-ресурсы:

1. Studref - Студенческие реферативные статьи и материалы (info{ат}studref.com) © 2017 - 2021
2.nsportal.ru - Образовательная социальная сеть.
3.Основы программирования на Python. Учебное пособие для СПО.-Электронная книга..,Дата выхода на ЛитРес: 01 сентября 2021 Дата написания: 2021,Объем: 287 стр
4.Kopilkaurokov.ru - сайт для учителей.

Глоссарий

1.Алгоритм - последовательность шагов, описывающая решение конкретной задачи или преобразование данных.

2.Программирование - процесс создания программного кода для автоматизации задач или выполнения определенных операций.

3.Язык программирования - формальный язык, используемый для написания компьютерных программ.

4.Инструкция - одна команда или операция в программе.

5.Переменная - имя, связанное с определенным значением или данными в программе.

6.Тип данных - спецификация, определяющая, какие виды данных может содержать переменная (например, целые числа, строки, булевы значения).

7.Цикл - конструкция в программе, позволяющая выполнять определенные действия многократно.

8.Условие - логическое выражение, которое определяет, какие действия выполнить в программе в зависимости от истинности выражения.

9.Функция - фрагмент кода, который может быть вызван для выполнения определенной операции.

10.Параметр функции - значение, передаваемое в функцию в момент вызова для использования внутри функции.

11.Массив (список) - структура данных, которая позволяет хранить множество значений под одним именем.

12.Рекурсия - техника, при которой функция вызывает саму себя для решения задачи.

13.Алгоритмическая сложность - мера того, насколько эффективно работает алгоритм при обработке данных или выполнении задачи.

14.Отладка - процесс поиска и исправления ошибок в программном коде.

15.Интерфейс пользователя (UI) - визуальные и функциональные элементы, с которыми взаимодействует пользователь при работе с программой.

16.Интеграция - процесс объединения различных компонентов программы или системы для их совместной работы.

17.API (Интерфейс программирования приложений) - набор функций и процедур, которые позволяют разным программам взаимодействовать друг с другом.

18.Подпрограмма - многократно используемый фрагмент кода, обычно выполняющий определенную задачу.

19.Псевдокод - подход к описанию алгоритмов, использующий смешанный набор натурального языка и кода для упрощения понимания.

20.Целостность данных - принцип обеспечения согласованности и надежности данных в программе или системе.
21.Python - популярный интерпретируемый язык программирования, известный своей простотой и читаемостью.
22.Интерпретатор Python - программа, выполняющая код Python, строка за строкой.
23.Значение None - специальное значение, представляющее отсутствие данных или ничего.
24.Комментарий - текст в коде, который игнорируется интерпретатором и используется для объяснения кода или оставления заметок.
25.Инструкция print() - инструкция для вывода данных на экран.
26.Переменная - место для хранения данных, которое можно изменить в ходе выполнения программы.
27.Тип данных - определяет, какие виды данных могут храниться в переменных (например, int, float, str).
28.Строка (string) - тип данных, представляющий текст, заключенный в кавычки.
29.Список (list) - структура данных, представляющая упорядоченную коллекцию элементов.
30.Словарь (dictionary) - структура данных, представляющая коллекцию пар "ключ-значение".
31.Условное выражение (if) - конструкция для выполнения кода в зависимости от условия.
32.Цикл for - конструкция для итерации по элементам последовательности (например, списка).
33.Цикл while - конструкция для выполнения кода, пока условие выполняется.
34.Функция - блок кода, который может быть вызван с определенными аргументами для выполнения определенной задачи.
35.Модуль - файл, содержащий Python-код и функции, который может быть импортирован в другие программы.
36.Исключение (exception) - ошибка, возникающая во время выполнения программы, которую можно обработать.
37.Итератор - объект, позволяющий итерировать по последовательности данных (например, for в цикле).
38.Индексация - обращение к элементу списка или строки по его порядковому номеру.
39.Метод - функция, связанная с объектом (например, методы строки str.upper() или список list.append()).

40.Модуль math - стандартный модуль Python, предоставляющий математические функции и константы.