



Министерство науки и высшего образования Российской Федерации
Рубцовский индустриальный институт (филиал)
ФГБОУ ВО «Алтайский государственный технический
университет им. И.И. Ползунова»

Л.А. ПОПОВА

ПРОГРАММИРОВАНИЕ

Учебно-методическое пособие для студентов первого курса
направления «Информатика и вычислительная техника»
очной и заочной форм обучения

Рубцовск 2021

ББК 32.973.2

Попова Л.А. Программирование: Учебно-методическое пособие для студентов первого курса направления «Информатика и вычислительная техника» очной и заочной форм обучения / Л.А. Попова. – Рубцовск: РИИ, 2021. – 94 с. [ЭР].

Рассмотрены основные понятия и принципы программирования. Приведены примеры решения задач по изложенному материалу. Прилагаются задания к практическим и лабораторным работам, а также вопросы для самостоятельной подготовки к текущему и промежуточному тестированию.

Учебно-методическое пособие предназначено для проведения практических и лабораторных работ по курсу «Программирование» у студентов направления 09.03.01 «Информатика и вычислительная техника» 1-го курса.

Рассмотрены и одобрены на заседании кафедры прикладной математики Рубцовского индустриального института.
Протокол № 9 от 18.03.2021 г.

Рецензент: начальник
Информационно-технического отдела РИИ

А.Н. Цыганков

© Рубцовский индустриальный институт, 2021

Содержание

ВВЕДЕНИЕ	4
1 ПОНЯТИЕ АЛГОРИТМА И ПРОГРАММИРОВАНИЕ.....	4
2 ЯЗЫКИ ПРОГРАММИРОВАНИЯ.....	8
3 СИНТАКСИС ЯЗЫКА.....	11
4 ОПЕРАТОРЫ И ВЫРАЖЕНИЯ	15
5 ФУНКЦИИ И МОДУЛИ.....	22
6 ОПЕРАТОРЫ УПРАВЛЕНИЯ ПРОГРАММНЫМ ПОТОКОМ.....	34
7 СТРУКТУРЫ ДАННЫХ.....	38
8 ФАЙЛЫ	50
9 ГРАФИЧЕСКИЕ ВОЗМОЖНОСТИ МОДУЛЯ TKINTER.....	58
10 ОСНОВЫ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ (ООП). КЛАССЫ И ОБЪЕКТЫ.....	63
11 ЛАБОРАТОРНЫЕ РАБОТЫ.....	66
12 ЗАДАНИЯ ДЛЯ ПРАКТИЧЕСКИХ ЗАНЯТИЙ И САМОСТОЯТЕЛЬНОЙ РАБОТЫ.....	88
13 ВОПРОСЫ ДЛЯ ПРОМЕЖУТОЧНОЙ АТТЕСТАЦИИ	93
СПИСОК ИСПОЛЬЗОВАННОЙ И РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ.....	93

ВВЕДЕНИЕ

Учебно-методическое пособие написано в соответствии с программой дисциплины «Программирование» для студентов первого курса направления «Информатика и вычислительная техника», предназначено для аудиторной и самостоятельной работы по данному курсу.

Пособие содержит теоретический материал по основам программирования, примеры решения задач, задания к лабораторным работам и практическим занятиям, а также вопросы для самостоятельной работы и подготовки к текущему контролю успеваемости и промежуточной аттестации.

1 ПОНЯТИЕ АЛГОРИТМА И ПРОГРАММИРОВАНИЕ

1.1 Алгоритм

Алгоритм – это система предписаний, определяющая процесс перехода от исходных данных к результату, предназначенная некоторому исполнителю для решения некоторого класса задач.

Команда – одно действие алгоритма. Каждый алгоритм должен быть составлен в *системе команд* некоторого *исполнителя*. Решения задач, содержащих переменные (*класса задач*), производят в общем виде.

Основные свойства алгоритма

Понятность – алгоритм должен включать только те команды, которые доступны исполнителю и входят в его систему команд. Каждая команда алгоритма должна быть четкой, однозначной.

Дискретность – алгоритм должен представлять процесс решения задачи как последовательное выполнение некоторых простых шагов. Выполнение очередного шага начинается после завершения предыдущего и реализуется за конечный отрезок времени.

Детерминированность (определенность) – в каждый момент времени следующий шаг работы однозначно определяется значениями данных, полученными на предыдущих шагах. Таким образом, алгоритм выдает один и тот же результат для одних и тех же исходных данных.

Результативность (разрешимость) – при корректно заданных исходных данных алгоритм должен завершать работу и выдавать результат за конечное число шагов.

Массовость – алгоритм составляется для целого класса задач и должен быть пригодным для разных наборов исходных данных.

Таким образом, алгоритм дает механическую инструкцию для решения задачи. В этом причина того, что алгоритмы могут быть реализованы на ЭВМ.

Основные способы записи алгоритмов:

- словесный (естественный язык);
- псевдокод;
- блок-схемы (графический способ);

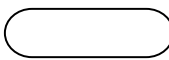
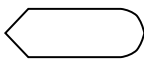
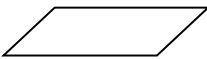

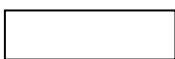

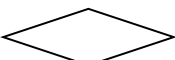


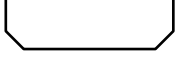
– язык программирования (искусственный язык).

Псевдокод – компактный (зачастую неформальный) язык описания алгоритмов, использующий ключевые слова языков программирования, но опускающий несущественные подробности и специфический синтаксис.

Псевдокод и словесный способ описания алгоритма широко используется в учебниках и научно-технических публикациях, а также на начальных стадиях разработки компьютерных программ.

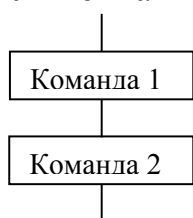
Блок-схема – это графическое изображение алгоритма в виде определенным образом связанных между собой нескольких типов блоков.

Графические элементы блок-схемы

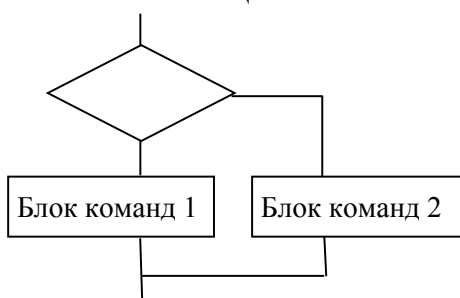
Основные блоки	Назначение	Вспомогательные блоки	Назначение
	Начало; конец		Терминал
	Ввод; вывод		Печатный документ
	Вычисления и присваивания		Магнитный диск
	Проверка условия		Соединение
	Начало цикла		
	Конец цикла		

Типы алгоритмических структур

Линейная



Разветвляющаяся



Циклическая



При составлении блок-схем следует руководствоваться *правилами*:

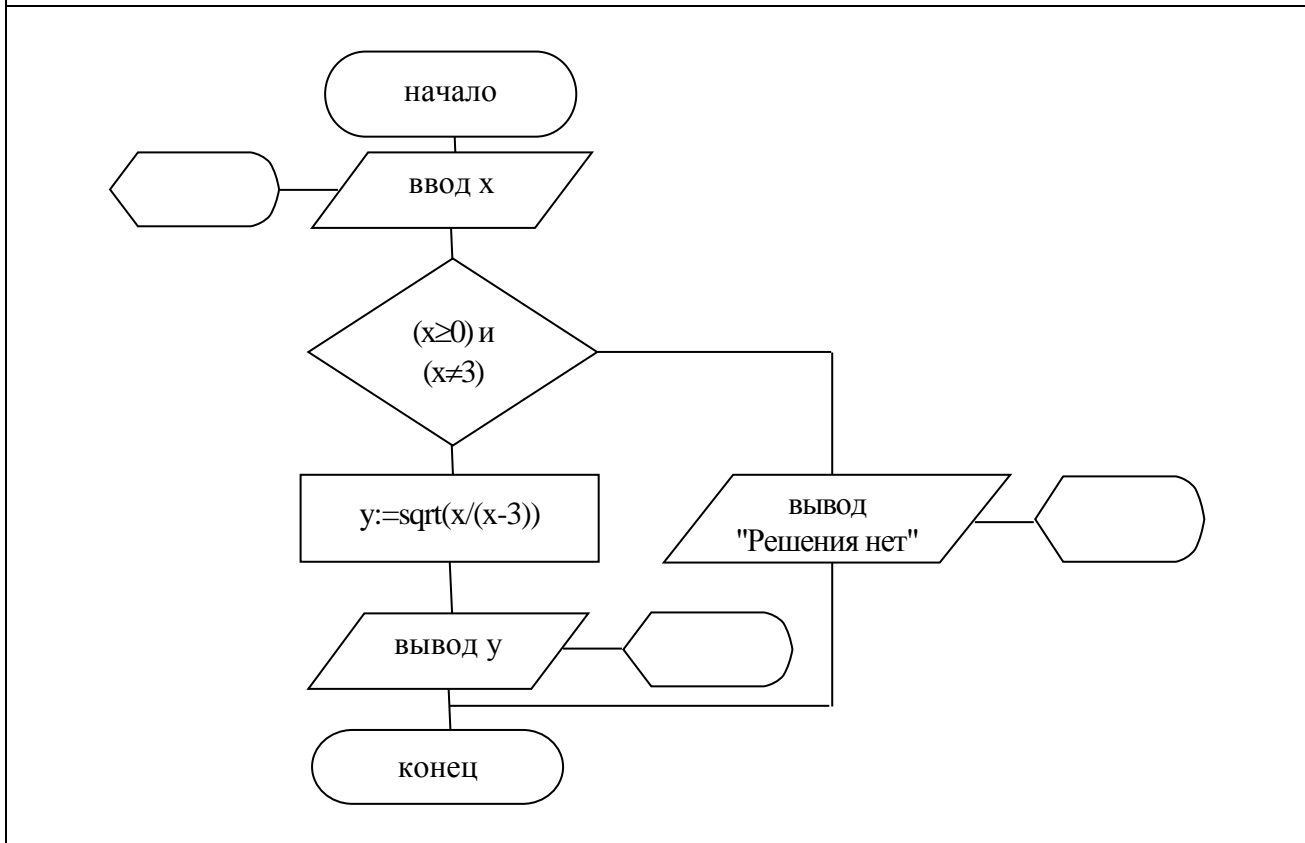
- блок-схема должна развиваться сверху вниз; при необходимости – слева направо;
- блок *Начало* имеет 1 выход, *Конец* – 1 вход; блоки ввода/вывода, вычисления и присваивания, начала и конца цикла имеют 1 вход и 1 выход; блок условия – 1 вход и 2 выхода;
- выход рисуется точно под входом.

Пример: Вычислить значение функции $y = \frac{\sqrt{x}}{x-3}$.

Словесный алгоритм	Псевдокод
--------------------	-----------

1 Ввести x	<u>начало</u>
2 Если $(x \geq 0)$ и $(x \neq 3)$	<u>ВВОД</u> x
2.1 $y = \sqrt{x}/(x-3)$	<u>если</u> $(x > 0)$ и $(x < 3)$
2.2 вывести y	<u>то</u> $y = \sqrt{x}/(x-3)$
2.3 перейти на п.4	<u>ВЫВОД</u> y
3 иначе	<u>иначе</u>
3.1 вывести "Решения нет"	<u>ВЫВОД</u> "Решения нет"
4 Завершить алгоритм	<u>конец</u>

Блок-схема



Языки программирования – это искусственно созданные языки, которые в отличие от естественных имеют ограниченное число "слов" и очень строгие правила записи команд и других конструкций языка.

1.2 Программирование

Программирование – это искусство получения ответов от машины.

Глобальная цель написания любой программы – это управление аппаратными ресурсами компьютера для реализации алгоритма решения задачи.

Компьютерная программа – это набор элементарных команд процессора, представленный в виде последовательности байтов машинного кода. Обычно программа пишется на одном из языков программирования в виде набора команд (алгоритма), который называется *исходным кодом* (*листингом*) программы. Затем с помощью специальной программы, называемой *транслятором*, переводится в набор инструкций процессора.

Трансляторы бывают двух типов:

- интерпретаторы – проводят покомандную обработку и выполнение исходной программы;
- компиляторы – преобразовать программу на машинный язык (в машинный код) после предварительной проверки ее синтаксиса.

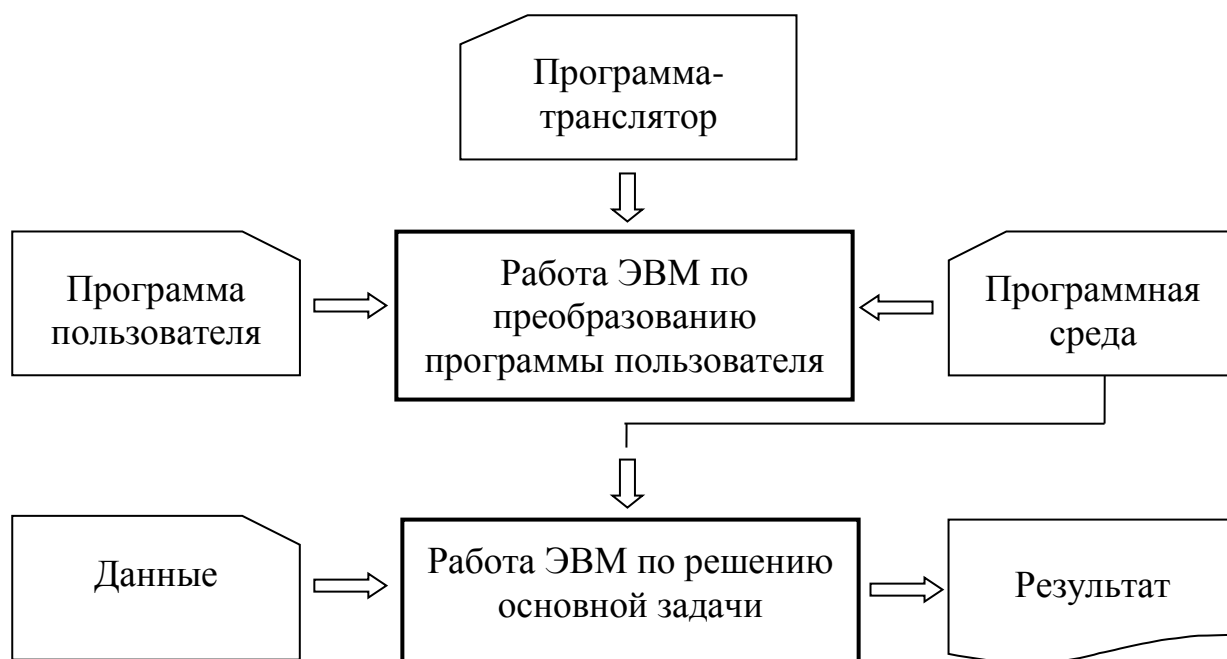


Рисунок 1 – Схема обработки программы пользователя на ЭВМ

Разработка программ выполняется в несколько этапов.

Анализ – разделение задачи на составные части (подзадачи); в структурном программировании реализуется подход «сверху-вниз». Разработка контрольных примеров.

Проектирование – составление формальной модели задачи, объектов и потоков информации. На этом этапе привлекаются специалисты и эксперты, хорошо знакомые с предметной областью, для которой составляется задача.

Создание (реализация) – деление программы на блоки и объединение их в единое решение задачи, написание программного кода.

Отладка (поиск логических ошибок в программе путем прогона ее на компьютере и их исправление) и *тестирование* (выполнение программы для различных наборов данных, реализация контрольных примеров).

Использование.

Поддержка (усовершенствование). Как говорят программисты, «Программы вырастают», а не строятся.

Решение любой задачи на ЭВМ состоит из следующих этапов:

1. Постановка задачи.
2. Формализация задачи.
3. Разработка контрольных примеров.
4. Построение алгоритма решения задачи.
5. Составление программы на языке программирования.
6. Отладка и тестирование программы.
7. Выполнение расчетов и анализ полученных результатов.

1.3 Вопросы для текущего контроля успеваемости

- 1) Что такое алгоритм? Кто (или что) может быть исполнителем алгоритма?
- 2) Перечислите и охарактеризуйте основные свойства алгоритма.
- 3) Какие используются основные способы записи алгоритма?
- 4) Какие существуют типы алгоритмических структур?
- 5) Что такое программа? В чем заключается основная цель написания компьютерной программы?
- 6) Какие программы называются трансляторами?
- 7) Опишите схему обработки программы пользователя на ЭВМ.
- 8) Перечислите и охарактеризуйте этапы разработки программы.

2 ЯЗЫКИ ПРОГРАММИРОВАНИЯ

2.1 Уровни языков программирования

Разные типы процессоров имеют разные наборы команд. Если язык программирования ориентирован на конкретный тип процессора и учитывает его особенности, то он называется языком *низкого уровня*. Его операторы близки к машинному коду. Этот язык дает полный контроль над аппаратным обеспечением компьютера и генерирует очень эффективный исполняемый код. Однако программирование требует больших затрат времени, и он труден для изучения. К языкам низкого уровня относятся машинные языки и языки символического кодирования (Автокод, *Ассемблер*).

Языки программирования, имитирующие естественные языки, обладающие укрупненными командами, ориентированными на решение прикладных содержательных задач, называются языками *высокого уровня*.

Языки программирования высокого уровня имеют ряд достоинств:

- набор операций, допустимых для использования, не зависит от набора машинных команд, а выбирается из соображений удобства реализации алгоритмов решения задач определенного типа;
- конструкции команд (операторов) задаются в удобном для человека виде;
- поддерживается широкий набор типов данных.

2.2 Поколения языков программирования

На самых первых машинах с программным управлением алгоритм решения задачи (программа) записывался на внутреннем языке машины, на языке двоичных кодов. Это значит, человек, готовивший программу для решения на ЭВМ, обдумывал каждое условие ветвления, все необходимые указания для ЭВМ, предусматривал конкретные ячейки памяти для хранения исходных данных, промежуточных и окончательных результатов, принимал решение о форме выдачи результата.

Программы писались на специальных бланках, после этого переносились на перфоленты или перфокарты и в таком виде передавались в вычислительный центр. Такой способ принято называть *ручным* программированием.

Второе поколение ознаменовалось появлением в начале 1950-х годов языка программирования Ассемблера. Вместо нулей и единиц использовались операторы, которые были похожи на слова английского языка. Компилятор преобразовывал эти выражения в машинные коды.

Третье поколение (относится к 1960-м годам) отмечено появлением первых языков программирования высокого уровня. Эти языки впервые позволили решать задачи из разных областей.

Четвертое поколение языков программирования зародилось в конце 1970-х прошлого века, развитие их продолжается по настоящее время. Языки постепенно становятся менее процедурными, в основе их построения лежит концепция объектно-ориентированного программирования (ООП), когда программный код становится более эффективным и от программиста требуются только указания того, что надо сделать, без подробного описания, как это должно быть выполнено.

2.3 Язык программирования Python

Python (Питон) – это простой в освоении и мощный язык программирования. Его разработку начал Гвидо Ван Россум (Guido Van Rossum) в 1990-е годы. Python активно развивается, новые версии (с добавлением/изменением языковых свойств) выходят примерно раз в два с половиной года.

Благодаря своей открытой природе, он был портирован на множество платформ (т.е. изменен таким образом, чтобы работать на них).

Python является интерпретируемым, что делает его идеальным языком для написания сценариев и быстрой разработки приложений в различных областях и на большинстве платформ. Программа просто выполняется из исходного текста. Python сам преобразует этот исходный текст в некоторую промежуточную форму, называемую байткодом, а затем переводит его на машинный язык и запускает.

Python предоставляет эффективные высокоуровневые структуры данных, а также простой, но эффективный подход к объектно-ориентированному программированию (ООП). Он предоставляет возможность работать с xml/html файлами, с http запросами, создавать веб-сценарии, выполнять действия с изображениями, аудио и видео файлами, используется в робототехнике. Python можно встраивать в программы на C/C++, чтобы предоставлять возможности написания сценариев их пользователям.

Разработчики языка Python придерживаются определенной философии программирования, называемой «The Zen of Python» («Дзен Пайтона»). Ее текст выдается интерпретатором Python по команде `import this` (работает один раз за сессию). Автором этой философии считается Тим Петерс (Tim Peters).

2.4 Среда программирования IDLE (Python)

Установку Python следует проводить с официального сайта <https://www.python.org/>

После установки будут доступны:

- интерактивная оболочка Python (предназначена для выполнения команд);
- IDLE (Integrated Development and Learning Environment) – интегрированная среда разработки и обучения на языке Python. Сочетает возможности интерактивного и файлового (пакетного) режимов обработки команд.

При работе в интерактивном режиме в начале строки выводятся >>> (три знака «больше»), которые называются приглашением, после них можно вводить команды (командная строка интерпретатора Python).

Если после приглашения ввести код и нажать клавишу Enter, например:

```
>>> 123*4567
```

```
>>> print("Привет, Мир!")
```

Python сразу же выведет результат работы.

При работе с командной оболочкой Python вернуться назад и что-то изменить нельзя. Однако можно легко повторить команду (при необходимости с изменениями): для этого следует клавишей «вверх» (или с помощью мыши) перейти к нужной команде и нажать клавишу Enter.

В интегрированную среду разработки IDLE входит текстовый редактор, который предназначен для ввода и выполнения сразу нескольких команд (программы). Программа сохраняется в отдельный файл, к которому можно многократно обращаться.

Одно из самых основных достоинств редактора – это подсветка синтаксиса, когда разные элементы программы имеют разные цвета в зависимости от их назначения.

Команда File → New File (или сочетание клавиш Ctrl+N) используется для создания нового файла. Открывается пустое окно с заголовком Untitled.

Команда File → Save – для сохранения файла. После внесения изменений файлы необходимо снова сохранять, так как редактор запускает на выполнение (Run → Run Module) только сохраненные файлы.

Run → Python Shell – открытие оболочки Python без запуска программы пользователя на выполнение.

Для быстрого получения информации о любой функции или операторе Python служит встроенная функция help. Это особенно удобно при использовании командной строки интерпретатора. Например, help(print) – вызов подсказки о функции print, help(help) – описание самой функции help.

Удобство работы в среде IDLE заключается в том, что можно проверять правильность работы отдельных команд в интерактивном режиме, копировать и сохранять последовательность команд в виде отдельного файла с помощью текстового редактора.

Примечание: в редакторе нельзя просто записывать выражения, можно выводить из значения, например, команда print(23*58) является правильной, 23*58 – неправильной.

Для выхода из среды используется команда File → Exit из главного меню, функция exit() в командной строке интерпретатора или нажатие кнопки закрытия окна.

2.5 Вопросы для текущего контроля успеваемости

- 1) В чем заключено основное отличие языков программирования низкого и высокого уровня?
- 2) Опишите поколения языков программирования, приведите примеры.
- 3) Приведите основные характеристики языка Python.
- 4) Вызовите «The Zen of Python» и постарайтесь понять смысл тезисов этой философии.
- 5) Для чего предназначен интерактивный режим? Как можно повторять команды?
- 6) Как выполняются действия: создание, открытие, сохранения программы в виде файла, запуск программы на выполнение?

3 СИНТАКСИС ЯЗЫКА

На первых этапах изучения языка программирования особое внимание следует уделять знанию синтаксиса (правил записи) и пониманию семантики (смысла) команд и других конструкций языка.

3.1 Идентификаторы

Алфавит языка состоит из символов Unicode (Юникода).

Идентификатор – это имя объекта (переменной, функции, класса и т.п.), используемое для его обозначения. При выборе имен объектов необходимо соблюдать следующие правила:

- первым символом идентификатора должна быть буква из алфавита (английского, русского или другого) или символ подчеркивания (`_`);
- остальная часть идентификатора может состоять также из букв, знаков подчеркивания или цифр (0-9).

Примеры допустимых идентификаторов: *a*, *__my_name*, *name_23*, *a1b2_c3* и символы *utf8_*.

Примеры недопустимых имен идентификаторов: *2things*, *пробелы не допустимы*, *a1_b2>c3* и *"текст в кавычках"*.

Идентификаторы чувствительны к регистру. Например, *myname* и *MyName* – это разные имена.

3.2 Физические и логические строки

Исходный текст программы состоит из физических и логических строк. Физическая строка заканчивается нажатием Enter. Логическая строка – это то, что Python воспринимает, как единое целое.

Например, фрагмент кода:

```
>>> i = 5
>>> print(i)
```

состоит из двух физических и двух логических строк.

А фрагмент:

```
>>> i = 5; print(i)
```

на одной физической строке содержатся две логические строки.

Общепринятое правило: одной физической строке соответствует не более одной логической строки.

Пример написания одной логической строки, занимающей несколько физических строк (явное объединение строк):

```
>>> s = 'Это строка. \
        Строка продолжается.'
```

```
>>> print(s)
```

Выведет результат:

```
'Это строка. Строка продолжается.'
```

Использование обратной косой черты (называется экранирование) не обязательно, когда в логической строке есть открывающаяся круглая, квадратная или фигурная скобка, но нет закрывающейся (неявное объединение строк).

Разделение физической строки на две (или более) логические строки следует использовать лишь в случае очень длинных строк.

3.3 Отступы

В Python наличие или отсутствие пробелов (отступов) в начале строки очень важно. Отступы (как правило, 4 пробела или табуляция) в начале логической строки используются для вложенных команд (например, при описании функций, классов, операторов управления).

Неправильные отступы могут приводить к возникновению ошибок. Например:

```
a = 5
print(a) # Ошибка! Пробел в начале строки
```

При запуске на выполнение этого фрагмента выведется соответствующее сообщение.

Не рекомендуется смешивать пробелы и символы табуляции в отступах в пределах одного файла, так как это не всегда работает корректно.

Подробную информацию о стиле кода в языке Python (PEP8) можно прочитать на сайте: <https://pep8.ru/doc/pep8/>

3.4 Комментарии

Программы необходимо снабжать комментариями для лучшего понимания программного кода.

Комментарии – текст, который пишется после символа #, и представляет интерес лишь как заметка для человека, читающего программу, интерпретатор их не воспринимает.

Пример комментариев:

```
# R – радиус описанной окружности
```

Текст программы отвечает на вопрос «как?», а комментарии должны объяснять «зачем?», «почему?» или «для чего предназначены?».

3.5 Константы

Строка – это некоторая последовательность символов. Строку можно задать в одинарных кавычках (апострофах) или в двойных кавычках.

Можно вывести любой символ из Unicode по его коду (формат символа: `\uXXXX`, X – цифра в шестнадцатеричной системе счисления: 0, 1, ..., 9, a, ..., f, A, ..., F).

Функция `ord(char)` возвращает десятичный код символа (`char`), а функция `chr(kod)` возвращает символ по его десятичному коду (`kod`).

Можно задавать «многострочные» строки с использованием тройных кавычек (`"""` или `'''`). В пределах тройных кавычек можно использовать текст, заключенный в кавычки или апострофы. Например:

```
"""Это многострочная строка. Это ее первая строка.
```

```
Это ее вторая строка.
```

```
"""Это продолжение строки в кавычках",
```

```
'продолжение строки в апострофах'.
```

```
''' #Здесь строка закончилась
```

Если расположить рядом две строковых константы, Python автоматически их объединит. Например, `'What's ' 'your name?'` автоматически преобразуется в `"What's your name?"`.

Строки *неизменяемы*, то есть после создания строки ее нельзя изменить.

Примеры строк в зависимости от интерпретации символов в ней:

`'\n\r\t\n\b\b'` – строка с экранированием (`\n` – перевод на другую строку),

`r'C:\temp\new'` – неформатированная («сырая») строка,

`b'bytes1+2'` – строка байтов для английских букв, цифр и некоторых специальных символов,

`'Байты'.encode('utf-8')` – строки байтов для кодировки utf-8,

```
# b'\xd0\x91\xd0\xb0\xd0\xb9\xd1\x82\xd1\x8b'
```

Преобразование к строке байтов можно выполнить с помощью функции:

```
bytes('bytes и байты 123', encoding = 'utf-8')
```

```
# b'bytes \xd0\xb8 \xd0\xb1\xd0\xb0\xd0\xb9\xd1\x82\xd1\x8b 123'
```

Числа в Python бывают трех типов: целые, с плавающей точкой и комплексные.

Примеры чисел:

– целые: 2, 12587, -1456;

– с плавающей точкой: 3.23, .2, -3.1e5 и 52.3E-4, где E – обозначение степени числа 10 (экспоненциальный или научный формат);

– комплексные: (-5 + 4j), (2.3 - 4.6j).

Целые числа могут быть произвольной длины. Например,

```
>>> 123456789*987654321
```

Целые числа могут быть представлены не только в десятичной системе счисления, но и в двоичной (`bin`), восьмеричной (`oct`) и шестнадцатеричной (`hex`) системах счисления. Например:

```
>>> bin(29) # '0b11101'
```

```
>>> oct(29) # '0o35'
```

```
>>> hex(29) # '0x1d'
```

Результат – строка. Однако можно выполнить обратное преобразование:

```
>>> int('0o35', 8) # 29
```

Преобразует строку '0o35' из системы счисления с основанием 8 в десятичную систему. При таком конвертировании основание системы может быть от 2 до 36 включительно, строка должна содержать только цифры, используемые в указанной системе счисления. Например,

```
>>> int('3gh', 20) # 1537
>>> 3*(20**2)+16*20+17 # 1537
```

По умолчанию все целые числа переводятся в десятичную систему счисления. Например:

```
>>> 0b111 # 7
>>> 0o23*3 # 57
```

Числа и строки называются *литеральными* константами (литеральный означает буквальный).

Значения некоторых (стандартных) констант хранятся в оперативной памяти (ОП) в течение всего времени работы программы (например, True, False, None, часто используемые натуральные числа).

3.6 Переменные

Переменная в программе – это способ обращения к участку внутренней (оперативной) памяти компьютера, в которой хранятся некоторые данные.

В программировании слово переменная обозначает именованное место для хранения данных (числа, текста, списка с числами или символами и т.д.). Также переменную можно рассматривать как *ярлык*, которым помечены некоторые данные.

При введении новой переменной в текст программы создается новая запись в соответствующей таблице, описывающей связи между именами переменных и адресами в ОП (таблица символов).

Переменной можно назначить пустое значение (None), которое говорит о том, что она ничего не содержит, но зарезервирована в программе. Например:

```
>>> my_var = None
>>> print(my_var)
None
```

Если переменная имеет значение, совпадающее со стандартной константой, то задается связь между этой переменной и соответствующим адресом ячейки ОП, содержащей константу. Например, если в программе использовать несколько переменных, имеющих значение None, то они все будут связаны с одним и тем же адресом в ОП.

Если значение переменной не совпадает со значением стандартной константы, то сначала определяется, где в ОП будет записано это значение, а затем создается связь, которая прописывается в таблице символов.

Проверить, совпадают или нет ссылки на переменные (или объекты) можно с помощью оператора is. Например:

```
>>> a = 1
>>> b = 1
>>> a is b # 1 является стандартной константой
True
```

```
>>> a = 1.
>>> b = 1.
>>> a is b # число с плавающей точкой 1.
           #НЕ является стандартной константой
```

False

Python все, что есть в программе, включая числа, строки и функции, рассматривает как объекты, имеющие следующие характеристики:

– *тип*. Например,

```
>>> type(1)
<class 'int'>
```

– *идентификатор* (адрес в оперативной памяти). Например,

```
>>> id(1) # 1853503744
```

– *размер*. Например,

```
>>> sys.getsizeof(1) # 14
# Более подробно о работе с функциями читайте
# в соответствующих разделах.
```

3.7 Вопросы для текущего контроля успеваемости

- 1) Для чего предназначены идентификаторы? Какие символы в них можно использовать?
- 2) Приведите примеры физических и логических строк. Какие правила их записи следует использовать?
- 3) Как записываются комментарии? Для чего они предназначены?
- 4) Приведите примеры констант различных типов.
- 5) С какой целью в программе используются переменные?
- 6) Что относится к объектам в Python? Какие общие характеристики они имеют? Приведите примеры.

4 ОПЕРАТОРЫ И ВЫРАЖЕНИЯ

Большинство предложений (логических строк) в программах содержат выражения. Простой пример выражения: $2 + 3$. Выражение состоит из операторов (в данном примере знак «+») и операндов (константы 2 и 3).

Оператор – это некоторое действия, которое может быть представлено в виде символа (+, *, =) или специального зарезервированного слова (and, or, not), выполняемое над данными – операндами (константами, переменными, функциями).

4.1 Операторы и их применение

Операторы языка Python описаны в таблице 1.

Таблица 1

<i>Оператор</i>	<i>Название</i>	<i>Примеры</i>	<i>Примечание</i>
+	Сложение	$3 + 5$ – результат 8; 'a' + 'b' – результат 'ab'	Суммирует числа или соединяет строки.

Продолжение таблицы 1

<i>Оператор</i>	<i>Название</i>	<i>Примеры</i>	<i>Примечание</i>
-	Вычитание	50 – 24 – результат 26; -5.2 – результат отрицательное число	Вычисляет разность двух чисел; если первый операнд отсутствует, он считается равным нулю.
*	Умножение	2 * 3 – результат 6; 'la' * 3 – результат 'lalala'; 4 * 'a' – результат 'aaaa'	Умножает числа или возвращает строку, повторенную заданное число раз.
**	Возведение в степень	3 ** 4 – результат 81	
/	Деление	4 / 3 – результат 1.3333333333333333	Результат – дробное число.
//	Целочисленное деление	4 // 3 – результат 1	Результат – целое число.
%	Остаток от деления	8 % 3 – результат 2; -25.5 % 2.25 – результат 1.5.	-25.5 = n * 2.25 + 1.5, где n = -12 – целое число.
<	Меньше	5 < 3 – результат False; 3 < 5 – результат True; 3 < 5 < 7 – результат True	Можно составлять произвольные цепочки сравнений. Результат – логического типа (False, True).
>	Больше	5 > 3 > 1 – результат True	
==	Равно	2 == 2. – результат True; 'str' == 'stR' – результат False	Проверяет, одинаковы ли значения у объектов. Результат – логического типа.
!=	Не равно	2 != 3 – результат True	Проверяет, верно ли, что объекты не равны.
<=	Меньше или равно	3 <= 6 – результат True	
>=	Больше или равно	3 >= 3 – результат True	
in	Проверка принадлежности	5 in range(5) – результат False 5 in range(6) – результат True	
is	Проверка тождественности	.1 is .1 – результат True 1 is .1 – результат False	
not	Логическое НЕ	not True – результат False not 0 – результат True	Применяется для данных разного типа. Результат – логического типа.
and	Логическое И	True and 0 – результат 0 False and 5 – результат False 5 and 10 – результат 10 (не False)	

<i>Оператор</i>	<i>Название</i>	<i>Примеры</i>	<i>Примечание</i>
or	Логическое ИЛИ	5 or 0 – результат 5 0 or False – результат False not 5 or True – результат True	
<<	Сдвиг влево	2 << 2 – результат 8	Сдвигает биты числа влево на заданное количество позиций.
>>	Сдвиг вправо	11 >> 1 – результат 5	Сдвигает биты числа вправо на заданное количество позиций.
&	Побитовое И	5 & 3 – результат 1	
	Побитовое ИЛИ	5 3 – результат 7	
^	Побитовое исключающее ИЛИ	5 ^ 3 – результат 6	
~	Побитовое НЕ	~5 – результат -6	Побитовая операция НЕ для числа x соответствует -(x+1)

4.2 Приоритет операторов

В таблице 2 представлен приоритет операторов от самого низкого до самого высокого (сверху вниз). Это означает, что в любом выражении Python сначала выполняет операторы, расположенные внизу таблицы, а затем операторы выше по таблице.

Таблица 2

<i>Оператор</i>	<i>Описание</i>
lambda	Лямбда-выражение
or	Логическое ИЛИ
and	Логическое И
not x	Логическое НЕ
in, not in	Проверка принадлежности
is, is not	Проверка тождественности
<, <=, >, >=, !=, ==	Сравнения
	Побитовое ИЛИ
^	Побитовое исключаяющее ИЛИ
&	Побитовое И
<<, >>	Сдвиги
+, -	Сложение и вычитание
*, /, //, %	Умножение, деление, целочисленное деление и остаток от деления
+x, -x	Положительное, отрицательное
~x	Побитовое НЕ
**	Возведение в степень

Операторы, имеющие одинаковый приоритет, при вычислении значения выражения выполняются слева направо (кроме присваивания).

4.3 Оператор присваивания

Общий вид оператора присваивания:

Переменная = выражение

Выполняется следующим образом:

- вычисляется значение выражения, заданного в правой части,
- затем результат записывается в ячейки оперативной памяти (или определяются адрес ячейки с полученным результатом) и создается ссылка от идентификатора-переменной на это значение.

Таким образом, тип переменной определяется по значению выражения, стоящего после знака равенства.

При записи выражений можно использовать конвертирование переменных (временно изменять тип переменной, если ее значение это позволяет сделать). Например:

```
a = '10'  
b = '8.25'  
c = int(a) * 2 + float(b)  
print(c, type(c))  
d = a * 2 + b  
print(d, type(d))
```

В результате переменной *c* будет присвоено значение 28.25, а переменной *d* значение '10108.25'.

Допускается использование нескольких знаков присваивания (=) в одном выражении. Например, $a = b = 5$. Присваивание выполняется справа налево.

Оператор присваивания предназначен для изменения значения переменной как без участия пользователя (см. операторы выше), так и при его непосредственном участии – вводе данных с клавиатуры:

```
name = input('Введите строку ')           # Ввод переменной строкового типа  
i = int(input('Введите целое число '))    # Ввод переменной целого типа  
f = float(input('Введите вещественное число '))
```

Присваивание вводимых с клавиатуры данных осуществляется с помощью функции `input()`, которая возвращает все символы из строки ввода до ближайшего символа "перевода строки" (`\n` – результата нажатия клавиши Enter).

Функция `input()` считывает все вводимые символы в одну строку, поэтому если несколько значений будут записаны на одной строке, то их сначала нужно разъединить (распарсить) с помощью метода `split()` с заданным разделителем. Если разделитель явно не задан, он по умолчанию деление строки на составные части выполняется по пробелам. Например:

```
name1, name2 = input().split() # Ввод значений двух строк через пробел
```

Если значение переменной переопределяется, то реализуется следующий алгоритм:

- 1) значение выражения (или константа) записывается в ОП;
- 2) создается связь между переменной и ее новым значением;
- 3) если больше нет переменных, ссылающихся на старое значение, то этот участок памяти помечается, как свободный.

Например:

```
>>> a = 500
```

```
>>> id(a) # 56454800
>>> a = 501
>>> id(a) # 56454576
>>> a = 500
>>> id(a) # 56454480
```

Данные в ОП, на которые нет ссылок, называются «мусор», поэтому для программ на языке Python реализован алгоритм «уборки мусора» (удаления ненужных данных из памяти), который выполняется без участия программиста и пользователя.

Если результат выполнения некоторой математической операции необходимо присвоить переменной, над которой эта операция производилась, то лучше использовать краткую форму записи оператора присваивания («переменная *оператор* = выражение»). Например:

```
a = 2
a = a * 3    # Обычная запись
a *= 3      # Краткая запись
```

4.4 Форматированный вывод данных

У функции вывода данных `print()` есть параметры, которые задаются по умолчанию, если пользователь явным образом не указал иное:

`sep=' '` (пробел) – разделитель между параметрами функции,

`end='\n'` (перевод строки) – символ который выводится после всех параметров строки.

```
>>> print(1, 2, 3, 4, 5)
1 2 3 4 5
>>> print(1, 2, 3, 4, 5, sep='_', end='!!!')
1_2_3_4_5!!!
```

Иногда возникают ситуации, когда нужно сконструировать строку, подставив в нее значения, полученные из ранее выполненных операторов, пользовательского ввода, файлов и т.д. При подстановке данных можно выполнить форматирование строк, которое реализуется с помощью оператора `%` или метода `format`.

4.4.1 Форматирование строк с помощью оператора `%`

Спецификаторы преобразования записываются в следующем порядке:

- 1) `%`
- 2) Ключ (опционально), определяет, какой аргумент из значения будет подставляться.
- 3) Флаги преобразования (таблица 4).
- 4) Минимальная ширина поля. Если `*`, значение берется из списка в скобках.
- 5) Точность, начинается с `'.'`, затем число (точность).
- 6) Модификатор длины (опционально).
- 7) Тип (таблица 3).

Форматы для вставки данных в строки

Формат	Результат
'%d', '%i', '%u'	Десятичное число.
'%o'	Число в восьмеричной системе счисления.
'%x', '%X'	Число в шестнадцатеричной системе счисления (буквы в нижнем или верхнем регистре).
'%e', '%E'	Число с плавающей точкой с экспонентой (экспонента в нижнем или верхнем регистре).
'%f', '%F'	Число с плавающей точкой (обычный формат).
'%g', '%G'	Число с плавающей точкой с экспонентой (экспонента в нижнем или верхнем регистре), если она меньше, чем -4 или точности, иначе обычный формат.
'%c'	Символ (конвертируется строка из одного символа или число – код символа).
'%r'	Строка (литерал Python – «сырая» строка).
'%s'	Строка (как обычно воспринимается пользователем).
'%%'	Знак '%'.

Если для подстановки требуется только один аргумент, то значением для подстановки будет сам аргумент (после символа %):

```
>>> 'Hello, %s!' % 'World' # 'Hello, World!'
```

А если нужно подставить несколько значений, то они объединяются в список и заключаются в скобки:

```
>>> '%d %s, %d %s' % (6, 'bananas', 10, 'lemons') # '6 bananas, 10 lemons'
```

```
>>> print("Number %.4f" % (1.33334)) # Number 1.3333
```

```
>>> print("%.4f, %.2f" % (1.33334, 153*0.43)) # 1.3333, 65.79
```

```
>>> print("%f, %f" % (1.33334, 153*0.43)) # 1.333340, 65.790000
```

```
>>> 153*0.43 # 65.78999999999999
```

Пример вывода символа по номеру (в десятичной системе счисления):

```
>>> '%c' % 50 # '2'
```

```
>>> '%c' % 189 # '½'
```

```
>>> '%c' % 999 # 'э'
```

Флаги преобразования

Флаг	Значение
#	Значение будет использовать альтернативную форму
0	Свободное место будет заполнено нулями
-	Свободное место будет заполнено пробелами справа
(пробел)	Свободное место будет заполнено пробелами слева
+	Свободное место будет заполнено пробелами слева (знак числа выводится)

```
>>> '%s' % 'Hello!' # 'Hello!'
```

```
>>> '%.2s' % 'Hello!' #усечение строки 'He'
```

```

>>> '%.*s' % (2, 'Hello!') # 'He'
>>> '%-10d' % 25 # '25      '
>>> '% 10d' % 25 # '      25'
>>> '%+10d' % 25 # '      +25'
>>> '%+10d' % -25 # '      -25'
>>> '%+10f' % 25 # '+25.000000'
>>> '%10.f' % 25 # '      25'
>>> '%#10.f' % 25 # '      25.'
>>> '%+10s' % 'Hello' # '  Hello'

```

Вставка в строку значений можно выполнять через ключ словаря. Например:

```

>>> '%(1)s, %(2).3f' % {'1': 'AB', '2': 2.33333} # 'AB, 2.333'

```

4.4.2 Метод format()

Строковый метод format() возвращает отформатированную версию строки, заменяя идентификаторы в фигурных скобках. Идентификаторы могут быть позиционными, числовыми индексами, ключами словарей, именами переменных.

Аргументов в format() может быть больше, чем идентификаторов в строке. В таком случае оставшиеся объекты игнорируются.

Идентификаторы могут быть либо индексами аргументов, либо ключами:

```

>>> "{}, {} and {}".format('one', 1, 'I') # 'one, 1 and I'
>>> "{1}, {2} and {0}".format('one', 1, 'I') # '1, I and one'
>>> nums = [3, 4, 5, 6, 2, 0]
>>> "{}{}{}{}".format(*nums) # '345'
>>> "{0}{2}{4}".format(*nums) # '352'
>>> u = {'name': 'Ivan', 'age': 18}
>>> '{name}-{age}'.format(**u) # 'Ivan-18'
>>> '{name}'.format(**u) # 'Ivan'
>>> '{name}-{age}'.format(name="pi", age=3.14) # 'pi-3.14'
>>> '{0}-{age}'.format("sin", **u) # 'sin-18'

```

Вывод атрибутов объекта:

```

>>> class house:
    size = "big"
    street = "main"
>>> h = house()
>>> '{0.size}, {0.street}'.format(h) # 'big, main'

```

Можно задавать ширину поля и выравнивание:

```

>>> '{name:10}-{age:3}'.format(**u) # 'Ivan      -18'
>>> '{name:>10}-{age:>3}'.format(**u) # '      Ivan-18'
>>> '{name:^10}-{age:^3}'.format(**u) # '      Ivan  -18 '

```

Вывод вещественных чисел:

```

>>> '{0}'.format(4/3) # '1.3333333333333333'
>>> '{0:f}'.format(4/3) # '1.333333'
>>> '{0:.2f}'.format(4/3) # '1.33'

```

```
>>> '{0:10.2f}'.format(4/3) # ' 1.33'  
>>> '{0:10e}'.format(4/3) # '1.333333e+00'
```

Выравнивание строк

```
>>> a = "Hello"  
>>> a.center(10) # ' Hello '  
>>> a.rjust(10) # ' Hello'  
>>> a.ljust(10) # 'Hello  '  
>>> a.ljust(10, '.') # 'Hello.....'  
>>> a.center(10, '.') # '..Hello...'  
>>> "%s" % (a.center(19)) # ' Hello  '
```

Подробную информацию о форматировании строк можно найти на сайте:
<https://pyformat.info/>

4.5 Вопросы для текущего контроля успеваемости

- 1) Что такое операторы? На какие группы их можно разделить?
- 2) Для чего задается приоритет операций? Как определить, в каком порядке будут выполняться операторы в выражении? Как этот порядок можно изменить? Приведите примеры.
- 3) Как записывается и реализуется оператор присваивания? Приведите примеры.
- 4) Для чего используется форматирование строк? Как оно реализуется?

5 ФУНКЦИИ И МОДУЛИ

5.1 Функции из стандартных модулей (библиотек)

Python распространяется с библиотекой стандартных модулей (более 200). Часть модулей встроена в интерпретатор по умолчанию, обеспечивая доступ к операциям. Например, `builtins` подключается к любой программе или интерактивному режиму автоматически при запуске самого Python и содержит описание стандартных типов (классов) `int`, `float`, ..., функций `min(...)`, `type(...)` и др., которые в свою очередь помещаются в таблицу встроенных имен. Подробную информацию можно получить, если вызвать список всех объектов (`dir`) или помощь (`help`).

```
>>> import builtins  
>>> dir(builtins)  
>>> help(builtins)
```

Модули встраиваются либо из соображений эффективности, либо для обеспечения доступа к примитивам операционной системы, например, модуль `sys`.

5.1.1 Операторы для импортирования модулей и функций

Подключить модуль к программе (или к интерактивному режиму) можно с помощью инструкции `import имя_модуля`. Тогда при обращении к его функциям и переменным каждый раз надо обязательно указывать имя модуля: *модуль.функция*.

Пример 5.1. Вычислить $y = \sqrt{x^6 + 1} - |2x - 3|$ при заданном значении x .

```
import math
x = float(input('x = '))
y = math.sqrt(x ** 6 + 1) - math.fabs(2 * x - 3)
print('y =', y)
```

Если название модуля слишком длинное или не удобно для использования, то для него можно создать псевдоним с помощью ключевого слова `as`. Например:

```
>>> import random as rnd
>>> rnd.random()
```

Чтобы импортировать функции в программу и при каждом обращении к ним не писать имя модуля, можно воспользоваться командой «`from модуль функция1, функция2`». Например:

```
from math import sqrt, fabs
x = float(input('x = '))
y = sqrt(x ** 6 + 1) - fabs(2 * x - 3)
print('y =', y)
```

Для импорта имен всех функций и переменных, использующихся в модуле, можно выполнить команду «`from модуль import *`». В общем случае такие записи лучше не использовать, чтобы избежать конфликтов между именами функций разных модулей.

5.1.2 Стек вызовов

Стек вызовов (машинный стек, стек исполнения) является неотъемлемой частью большинства языков программирования. *Стек* – это абстрактная структура данных, для которой доступны лишь две операции: добавить какой-либо элемент на вершину (`push` – положить в стек) и убрать верхний элемент (`pop` – забрать из стека).

Стек вызовов хранит функции, которые мы вызвали (явно или неявно). Когда при исполнении программы интерпретатор видит, что надо использовать какую-то функцию, то он кладет эту функцию на стек, если функция завершает свою работу, то он ее снимает со стека.

Важное свойство интерпретатора: всегда выполнять самую верхнюю функцию на стеке. После завершения ее выполнения, интерпретатор снимает функцию со стека.

Когда запускаем среду Python или программу пользователя, то внизу стека находится функция `module`, которая исполняет команды пользователя, описанные в основной части программы (`main`).

При вызове функции происходит ее добавление на стек вызовов, при этом размер стека увеличивается на 1, а снятие функции со стека уменьшает его размер на 1.

Например, пусть выполняется следующий код:

```
>>> import math
>>> x = -5
>>> y = 2.3
```

```
>>> print(abs(x - math.sqrt(y + math.log(y))))
```

При обращении к функции `log` она будет добавлена в стек вызовов, при этом его содержимое станет следующим:

<i>вершина стека</i>
<code>math.log</code>
<code>math.sqrt</code>
<code>abs</code>
<code>print</code>
<code>module.__main__</code>

Причем функция `math.log` будет выполняться, а все остальные ждать своей очереди.

5.1.3 Пространства имен

Пространство имен (`namespace`) определяет отображение имен в объекты, ему соответствует «таблица символов». Пространства имен создаются в различные моменты времени и имеют разную продолжительность жизни. Пространство имен, содержащее встроенные имена, создается при запуске интерпретатора и существует до завершения его работы (если их явным образом не изменять и не удалять). Глобальное пространство имен модуля `main` содержит в себе те имена, которые были объявлены на самом верхнем уровне кода. Оно создается, когда считывается код модуля, и, обычно, также существует до завершения работы интерпретатора.

Тело функции исполняется, не в то время, когда функцию определяют, а когда ее вызывают. При каждом вызове функции создается локальное пространство имен, необходимость создавать которого тесно связана со стеком вызовов и с так называемыми локальными переменными. Когда функция завершает свое исполнение, она не только снимается со стека, но и разрушает локальное пространство имен, которое было создано.

При объявлении идентификатора в программе (связи его с некоторым объектом) он автоматически добавляется в соответствующую ему таблицу символов и существует до тех пор, пока не будет завершена работа функции (и разрушено пространство имен) или идентификатор не будет удален командой `del` (например, `del x`).

5.2 Описание функций пользователя

Функция – это объект, в тексте программы представленный в виде последовательности команд и имеющий собственное имя.

Назначение функций:

- 1) многократное выполнение некоторого кода программы;
- 2) структурирование программы;
- 3) сокрытие деталей реализации (функцию можно сравнить с «черным ящиком»).

Имя функции хранится в памяти. После вызова функции задаются значения ее локальных переменных (описанных внутри тела функции). Это

приводит к созданию новой таблицы символов, использующейся для хранения локальных переменных функции.

Функции определяются при помощи зарезервированного слова `def` (от слова `define`). После этого слова указывается имя функции (задаваемое программистом), за которым следует пара скобок, в них можно указать имена некоторых переменных (*формальных параметров*), и заключительное двоеточие в конце строки. Далее следует блок команд (с одинаковым отступом), составляющих тело функции. Например:

```
def say (x):  
    # Блок, принадлежащий функции  
    print('Hello, %s!' %x)  
    # Конец функции  
say('World')      # Вызов функции  
name = 'Ivan'  
say(name)         # Еще один вызов функции
```

При определении функции ее имя также помещается в текущую таблицу символов. Тип значения, связанного с именем функции, распознается интерпретатором как функция, определенная пользователем (*user-defined function*). Например, сравните результаты выполнения команд:

```
>>> type(say)  
<class 'function'>  
>>> type(min)  
<class 'builtin_function_or_method'>  
>>> type(sys.getsizeof)  
<class 'builtin_function_or_method'>
```

Имена, указанные при описании функции, называются формальными параметрами, а значения, которые передаются функции при ее вызове, – *фактическими* параметрами (или *аргументами*). Можно вызывать одну и ту же функцию много раз, задавая различные входные данные в качестве аргументов.

Фактические параметры при вызове функции помещаются в локальную таблицу символов вызванной функции; в результате аргументы передаются через вызов по значению (*call by value*) (где значение – это всегда ссылка (*reference*) на объект, а не значение его самого). Если одна функция вызывает другую – то для этого вызова создается новая локальная таблица символов.

При обнаружении ссылки на переменную, в первую очередь просматривается локальная таблица символов, затем локальная таблица символов для окружающих функций, затем глобальная таблица символов и, наконец, таблица встроенных имен. Таким образом, глобальным переменным невозможно прямо присвоить значения внутри функций (если они конечно не упомянуты в операторе `global`) несмотря на то, что ссылки на них могут использоваться.

Например, описание и вызов функции:

```
def comp(a, b):    # Описание функции, a и b – формальные параметры  
    print('Произведение равно', a * b)  
comp(3, 4)       # Прямая передача значений, 3 и 4 фактические параметры
```

```
x = 5
y = 7 + 2 * x      # Объявление глобальных переменных x и y
comp(x, y)         # Передача переменных x и y в качестве аргументов
comp(5, 7 + 2 * 5)
```

Определили функцию с именем *comp*, которая использует два формальных параметра с именами *a* и *b*. При первом вызове функции *comp* в качестве аргументов передаются числа 3 и 4. Во втором случае функции передаются значения переменных *x*, *y*. А в третьем случае аргументами функции являются выражения.

Оператор *pass* используется в Python для обозначения пустого блока команд. Например:

```
def someFunction():
    pass
```

Применяется для резервирования имени функции и последующего описания ее тела.

Пример 5.2. Использование функции *map* из модуля *builtins*:

а) для ввода нескольких данных:

```
a, b, c = map(int, input('Введите три целых числа через пробелы: ').split())
```

б) для преобразования данных в соответствие с заданной функцией:

```
def func(a):
    return int(a) ** 2
```

```
s = '1 2 3'
```

```
a, b, c = map(func, s.split())
```

5.3 Область видимости объектов

Область видимости (*scope*) – фрагмент программы, в котором пространство имен непосредственно доступно (объявлено в этом блоке). Говорят, что идентификатор объекта «виден» в определенном месте программы, если в данном месте по нему можно обратиться к данному объекту. За пределами области видимости тот же самый идентификатор может быть связан с другой переменной или функцией, либо быть свободным (не связанным ни с каким объектом).

Область видимости любой переменной ограничена блоком, в котором она объявлена, начиная с места первого указания ее имени. Областью видимости функции будет ее тело.

Области видимости статичны, зависят от кода, который написали, и никак не зависят от его исполнения. Однако в процесс исполнения кода областям видимости соответствуют локальные пространства имен, которые созданы при вызове функции. Например, глобальному *scope* соответствует глобальный *namespace*. Области видимости функции будут соответствовать ее локальный *namespace*, когда мы ее вызовем. Таким образом, перекрывается весь программный код областями видимости, одни из которых будут включать в себя другие, и не будет зависеть от исполнения программного кода.

При вызове какой-либо функции и обращении к идентификатору выполняется обход по областям видимости (до первого встретившегося присваивания значения идентификатору) в следующем порядке:

- локальная область (local scope) – тело функции до текущего оператора;
- закрывающие области (enclosing scope), в которых данная функция определена (может отсутствовать) по иерархии снизу вверх;
- глобальная область (global scope) – головная часть программы;
- scope для builtins.

Таким образом, поиск переменной осуществляется по пространствам имен, которые связаны с соответствующими им областями видимости.

5.3.1 Локальные переменные

При описании функции можно задавать локальные переменные. Переменные, созданные в теле функции, нельзя использовать после того, как эта функция завершит работу, так как область их видимости ограничена функцией. Список локальных объектов можно вызвать с помощью функции `locals()`. Например:

```
x = 50 # x в пространстве имен main (1)
print('Объявили x =', x)
def func(x): # x в пространстве имен func (2)
    print('Передали в функцию аргумент x =', x) # x вызвали из func (2)
    x = 2 # x переопределили в пространстве имен func (3)
    print('Внутри функции локальный x =', x) # x вызвали из func (3)
func(x) # x передали в функцию из main (1)
print('После завершения работы функции x =', x) # x вызвали из main (1)
```

Результат работы:

```
Объявили x = 50
Передали в функцию аргумент x = 50
Внутри функции локальный x = 2
После завершения работы функции x = 50
```

Для того чтобы не было путаницы при работе функции, глобальным и локальным переменным, а также формальным параметрам надо задавать различные имена.

Переменные, созданные вне тела функции, будут видны и в самой функции. Например, глобальный `score` перекрывает весь программный код:

```
c = 30 # c в пространстве имен main
def composition():
    a = 10
    b = 20 # a, b в пространстве имен composition
    print('Произведение равно', a * b * c)
composition()
```

Результат работы:

```
Произведение равно 6000
```

Переменная `c` видна в теле функции `composition()`.

5.3.2 Глобальные переменные внутри функции

Чтобы присвоить некоторое значение переменной, определенной на высшем уровне программы (в *global scope*), необходимо указать, что ее имя не локально, а *глобально* (*global*). Без применения этого слова невозможно изменить значение глобальной переменной внутри тела функции. Например:

```
x = 50 # x в пространстве имен main (1)
print('Объявили x =', x)
def func():
    global x # x в пространстве имен main
    print('Передали в функцию глобальный x =', x)
    x = 2 # x переопределили в пространстве имен main (2)
    print('Изменили глобальное значение x на', x) # x вызвали из main (2)
func()
print('После завершения работы функции x =', x) # x вызвали из main (2)
```

Результат работы:

```
Объявили x = 50
Передали в функцию глобальный x = 50
Изменили глобальное значение x на 2
После завершения работы функции x = 2
```

Примечания:

- 1) Для того чтобы в теле функции сделать переменную глобальной, ее нужно объявить в глобальном пространстве имен до вызова функции.
- 2) Можно объявить сразу несколько глобальных переменных в теле функции, например, `global x, y, z`.

5.3.3 Нелокальные переменные

Есть еще один тип области видимости, называемый «*нелокальной*» (*nonlocal*) областью видимости, который используется для изменения значений переменных из *enclosing scope*. Они встречаются, когда определяют функции внутри других функций. Например:

```
x=50 # x в пространстве имен main (1)
print('Объявили x =', x)
def func1():
    x = 2 # x в пространстве имен func1 (2)
    print('Внутри функции func1 локальный x =', x)
    def func2():
        nonlocal x # x в пространстве имен func1 (2)
        x = 10 # x переопределили в func1 (3)
    func2()
    print('После вызова func2 локальный x изменился на', x)
func1()
print('после завершения работы func1 x =', x) # x вызвали main (1)
```

Результат работы:

```
Объявили x = 50
Внутри функции func1 локальный x = 2
```

После вызова func2 локальный x изменился на 10
после завершения работы func1 x = 50

Задание: замените «nonlocal x» на «global x», а затем удалить это зарезервированное слово, сравните результаты и сделайте выводы.

5.4 Способы указания параметров

5.4.1 Значения по умолчанию

Часть параметров функции может быть необязательной, и для нее будут использоваться значения по умолчанию, если при вызове функции явным образом не будут указаны другие значения. Значение по умолчанию должно быть константой и присвоено идентификатору в объявлении параметров функции. Например:

```
def say(message, number = 1):  
    print(message * number)  
say('Привет')  
say('Мир', 3)
```

Результаты работы:

```
Привет  
МирМирМир
```

Значениями по умолчанию могут быть только параметры, находящиеся в конце списка формальных параметров. Это связано с тем, что значения присваиваются параметрам в соответствии с их положением (*позиционные параметры*). Например, описание def func(a, b = 5) допустимо, а def func(a = 5, b) – не допустимо.

5.4.2 Ключевые параметры

Если имеется некоторая функция с большим числом параметров, и при ее вызове требуется указать только некоторые из них, то значения этих параметров можно задавать по их имени (*ключевые параметры*). В этом случае для передачи аргументов функции используется имя (*ключ*) вместо позиции параметра.

Есть два преимущества такого подхода: во-первых, использование функции становится легче, так как нет необходимости отслеживать порядок аргументов; во-вторых, можно задавать значения только некоторых аргументов, при условии, что остальные имеют значения по умолчанию. Например:

```
def func(a, b = 5, c = 10):  
    print('a = %d, b = %d, c = %d' % (a, b, c))  
func(3, 7) # a = 3, b = 7, значение по умолчанию c = 10  
func(25, c = 24)  
func(c = 50, a = 100)
```

5.4.3 Переменное число параметров

Иногда бывает нужно определить функцию, имеющую произвольное число параметров. Для этого можно использовать кортежи (перед ним ставится

одна *) или словари (перед ним ставится две *). Например:

```
def total(initial = 0, *numbers, **keywords):
    count = initial
    for number in numbers:
        count += number
    for key in keywords:
        count += keywords[key]
    print(count)
total(10, 1, 2, 3, vegetables = 50, fruits = 100)
```

Результат работы: 166

Когда объявлен параметр со звездочкой (например, *numbers), все позиционные аргументы, начиная с этой позиции и до конца, будут собраны в кортеж под именем numbers.

Аналогично, когда объявлен параметр с двумя звездочками (**keywords), все ключевые аргументы, начиная с этой позиции и до конца, будут собраны в словарь под именем keywords.

5.4.4 Только ключевые параметры

Если некоторые ключевые параметры должны быть доступны только по ключу, а не как позиционные, их можно объявить после параметра со звездочкой.

Пример:

```
def total(initial=0, *numbers, extra_number):
    count = initial
    for number in numbers:
        count += number
    count += extra_number
    print(count)
total(10, 1, 2, 3, extra_number = 50)
total(10, 1, 2, 3)
```

Вызовет ошибку, поскольку не указали значение аргумента для 'extra_number'.

В общем виде описание функции выглядит следующим образом:

```
def function_name([positional_parameters,
                  [positional_parameters_with_default,
                   [*pos_parameters_name,
                    [keyword_only_parameters,
                     [**kw_parameters_name]]]]]):
    # Function body
```

5.5 Оператор return

Оператор return используется для завершения функции и перехода из нее на оператор, следующий за точкой вызова. При этом функция может вернуть некоторое значение. После выполнения первого встретившегося оператора return функция завершает свою работу. Например:

```
def maximum(x, y):
    if x > y:
        return x
    else:
        return y
```

```
print(maximum(2, 3))
```

Функция `maximum` возвращает максимальный из двух параметров, которые передаются ей при вызове.

Оператор `return` без указания возвращаемого значения эквивалентен выражению `return None`.

Каждая функция содержит в неявной форме оператор `return None` в конце, если оператор `return` не указан явно.

5.6 Строки документации

Python имеет возможность использования некоторой справочной информации (строки документации – `docstrings`), которую можно получить из функции при разработке или выполнении программы. Строки документации записываются сразу после объявления функции (по правилам работы со строковыми константами, если «многострочная» строка, то в тройных кавычках). Например:

```
def printMax(x, y):
```

```
    """Выводит максимальное из двух чисел.
```

```
    Оба значения должны быть целыми числами."""
```

```
    if x > y:
```

```
        print(x, 'наибольшее')
```

```
    else:
```

```
        print(y, 'наибольшее')
```

```
printMax(3, 5)
```

```
print(printMax.__doc__) # __doc__ вызов строк документации функции printMax
    Результат работы:
```

```
5 наибольшее
```

```
Выводит максимальное из двух чисел.
```

```
Оба значения должны быть целыми числами.
```

5.7 Рекурсивные функции

Функция может вызывать другую функцию, а также может вызывать и саму себя. Рассмотрим это на примере функции вычисления факториала.

$0! = 1, 1! = 1.$

$n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot n = (n - 1)! \cdot n = (n - 2)! \cdot (n - 1) \cdot n$

Программный код на Python:

```
def factorial(n):
```

```
    if n == 0:
```

```
        return 1
```

```
    else:
```

```
        return n * factorial(n - 1)
```

```
print(factorial(3))
```

При вызове функции `print(factorial(3))` создается следующая структура:

Стек вызовов	Переменная в пространстве имен	Значение переменной	Результат функции
factorial	n	0	1
factorial	n	1	1*1=1
factorial	n	2	1*2=2
factorial	n	3	2*3=6
print			6
main			

Добавление функций на стек выполняется снизу вверх, а результат вычислений формируется при снятии функций со стека (сверху вниз).

Рекурсивные функции являются мощным механизмом в программировании. Однако они не всегда эффективны. Например, рекурсия:

```
def F(n):  
    if n > 2:  
        return F(n-1) + F(n-2)  
    else:  
        return 1
```

Имеет показательную сложность ($y=2^n$). Это означает, то при увеличении входного параметра на 1 время выполнения функции увеличивается примерно в 2 раза.

При разработке рекурсивной функции необходимо, прежде всего, проверять условия завершения рекурсии. Наиболее распространенной ошибкой является бесконечная рекурсия, когда цепочка вызовов функций продолжается до тех пор, пока не закончится свободная оперативная память в компьютере. Пример бесконечной рекурсии:

```
def hello_print():  
    print('Hello, World!')  
    hello_print()
```

5.8 Инструкция lambda

С помощью инструкции `lambda` создаются «анонимные» функции, так как их не обязательно ставить в соответствие некоторому идентификатору, как это делается с помощью инструкции `def`. Анонимные функции могут содержать лишь одно выражение, которое возвращается (без использования инструкции `return`). Выполняются они быстрее. Например:

```
>>> func = lambda x, y: x + y  
>>> func(1, 2) # 3  
>>> func('a', 'b') # 'ab'  
>>> (lambda x, y: x + y)(1, 2) # 3  
>>> func = lambda *args: args  
>>> func(1, 2, 3, 4) # (1, 2, 3, 4)  
>>> c = lambda x, y: x if x < y else y  
>>> c(5,6) # 5
```


5.9 Модули пользователя

Любая программа с расширением `.py`, содержащая функции и глобальные переменные, является модулем. Ее можно подключать к другой программе.

Модули должны быть сохранены в текущей папке или в папке с программой Python.

Модуль работает по-разному в зависимости от того, используется он сам по себе или импортируется в другую программу. У каждого модуля есть имя, и команды в модуле «знают» это имя. Этим свойством пользуются, когда необходимо узнать, запущен модуль как самостоятельная программа или импортирован. Для этого применяется атрибут модуля под названием `__name__`. Если он равен `'__main__'`, то программа запущена самостоятельно пользователем, иначе импортирована. Например, файл программы под именем `p1.py` содержит текст:

```
def hello():
    print('Привет!')
if __name__ == '__main__':
    print('Запущена самостоятельная программа')
else:
    print('Программу импортировали как модуль в другую программу')
```

Если запустить программу `p1.py`, то будет выведено:

Запущена самостоятельная программа

Файл программы под именем `p2.py` содержит текст:

```
import p1
```

...

Если запустить программу `p2.py`, то будет выведено:

Программу импортировали как модуль в другую программу

5.10 Пакеты программ

Пакеты – это удобный способ иерархически организовать модули. Такое часто встречается в стандартной библиотеке.

Пакет – это папка с модулями и специальным файлом `__init__.py`, который показывает Python, что эта папка особая, так как содержит модули Python. Например:

`sborka` – папка, в которой содержатся файлы:

```
__init__.py
```

```
summa.py
```

```
product.py
```

Содержимое файлов:

```
summa.py
```

```
def summ(a,b):
```

```
    """ Сумма двух аргументов
```

```
    Тип данных – любой"""
```

```
    return a + b
```

```
def summ_sqr(a,b):
    """ Сумма квадратов двух чисел
    Тип данных – целые или вещественные числа """
    return a**2 + b**2
```

```
division.py
def div(a,b):
    """Деление двух чисел
    Тип данных – числовой, делитель не равен 0 """
    return a / b
```

Код основной программы:

```
import sborka
x=5
y=10
import sborka.summa as s
print(s.summ(x, y))
from sborka import division
print(division. div(x, y))
help(s)
help(division. div)
```

5.11 Вопросы для текущего контроля успеваемости

- 1) Что такое стандартный модуль (библиотека)? Как выполняется вызов функции из него?
- 2) Для чего предназначен стек вызовов? В каком порядке функции загружаются в него?
- 3) Перечислите назначение и правила описания функций пользователя.
- 4) Что такое пространство имен (namespace)? На каком этапе идентификаторы добавляются в пространство имен? Где и как они прописываются?
- 5) Что такое области видимости? Как они связаны пространствами имен?
- 6) Чем отличаются глобальные, локальные и нелокальные переменные?
- 7) Перечислите способы указания параметров.
- 8) Для чего предназначен оператор return? В каких случаях его не надо использовать?
- 9) Приведите пример рекурсивной функции и опишите порядок ее выполнения.
- 10) Приведите пример функции с инструкцией lambda.
- 11) С какой целью создаются модули пользователя и пакеты программ?

6 ОПЕРАТОРЫ УПРАВЛЕНИЯ ПРОГРАММНЫМ ПОТОКОМ

В Python есть три оператора управления потоком: условный (if) и два циклических (for и while).

6.1 Оператор if

Условный оператор предназначен для записи разветвляющихся алгоритмов. Оператор if используется для проверки условий и выполнения некоторых действий в зависимости от истинностного значения проверяемого условия. Минимальная корректная запись оператора:

if условие:

оператор

Условие может быть простым или составным. В случае составного условия используются логические операторы: not, and, or.

В состав условного оператора может входить так называемый «else-блок». Если условие верно, выполняется блок выражений («if-блок»), иначе выполняется другой блок выражений («else-блок»).

Пример: Вычислить значение функции $y = \begin{cases} a + 6x, & x < a, \\ ax + 3x^2, & x \geq a, \end{cases}$

при заданных значениях константы $a = 1$ и переменной x (аргумента).

```
a = 1
```

```
x = float(input('Введите x='))
```

```
if x < a:
```

```
    y = a + 6 * x
```

```
else:
```

```
    y = a * x + 3 * x * x
```

```
print('y =', y)
```

Условные операторы могут быть вложенными. Однако если надо проверить несколько условий в одном операторе, то используется часть elif (которая объединяет части else и if). То есть вместо двух вложенных условных операторов «if ... else-if ... else» используется один «if ... elif ... else». Это облегчает чтение программы, а также не требует дополнительных отступов.

Логические строки оператора со служебными словами if, elif и else записываются на одном уровне, в конце этих строк ставятся двоеточия, за ними следуют блоки команд (с дополнительным отступом).

Пример 6.1. По введенному с клавиатуры цвету светофора вывести действие для водителя.

```
color = input('Введите цвет светофора: ')
```

```
if color == 'красный' or color == 'Красный':
```

```
    print('Остановись!')
```

```
elif color == 'желтый' or color == 'Желтый':
```

```
    print('Приготовься!')
```

```
elif color == 'зеленый' or color == 'Зеленый':
```

```
    print('Продолжай движение!')
```

```
else:
```

```
    print('Цвет введен неправильно.')
```

Пример 6.2. Решить линейное уравнение вида $ax = b$.

```
def line_ur(a, b):
```

```

if a == b == 0:
    return '(-inf, inf)'
elif a == 0 and b:
    return 'нет решений'
else:
    return '%.3f' %(b/a)
c, d = map(float, input('Введите два числа: ').split())
print(line_ur(c, d))

```

6.2 Цикл for

Оператор `for ... in` является оператором цикла, который осуществляет итерацию по последовательности объектов, т.е. принимает значение каждого элемента в заданном наборе. В цикле может использоваться необязательная часть *else*, которой завершается работа цикла.

Примеры использования цикла:

```

1) s = 0
   a, b, c = map(int, input('vv 3 numbers: ').split())
   for i in a, b, c:
       print(i)
       s += i

```

```

2) for a in map(int, input().split()):
    print(a**2)

```

```

3) s = 0
   n = int(input('Введите количество чисел: '))
   print('Введите числа через пробелы ')
   for i in range(n):
       s += int(input())
   else:
       print(s)

```

Если необходимо выполнить некоторые действия для переменной, изменяющейся на отрезке целых чисел от a до b с шагом c , то используется функция `range(a, b+1, c)`.

```

4) s = 0
   for i in range(a, b, c):
       print(i)
       s += i

```

```

5) for i in range(-5, 1, 2):
    print(i)

```

```

6) for i in range(min(a, b), max(a, b), c):
    print(i)
    s += i

```

Пример 6.3. Вычислить сумму $S = \sum_{i=1}^n \frac{1}{i^2}$.

```

n = int(input('n = '))
S = 0

```

```
for i in range(1, n+1):
    S += 1/(i**2)
print('S =',S)
```

6.3 Цикл while

Цикл `while` позволяет многократно выполнять блок команд, пока выполняется некоторое условие. Он также может иметь необязательную часть *else*. Например, определить машинную точность:

```
n = 1
k = 0
while n:
    n /= 10
    k += 1
print('Машинная точность составляет: 1e-%i' %k)
```

Результат работы:

Машинная точность составляет: 1e-324

Это означает, что на языке Питон можно выполнять вычисления с точностью больше, чем $1 \cdot 10^{-324}$ (это число воспринимается как 0).

При составлении программ с использованием цикла `while` надо предусмотреть условия, при которых: 1) цикл начинает выполняться, 2) завершает работу и 3) происходят изменения внутри тела цикла

6.4 Оператор break

Оператор `break` служит для прерывания цикла, т.е. прекращения выполнения команд, даже если условие цикла еще не приняло значение `False` или последовательность элементов не закончилась.

Если циклы `for` или `while` прервать оператором `break`, соответствующие им блоки `else` выполняться не будут. Например:

```
S = k = 0
print('Введите числа в столбик (условие завершения: неположительное число)')
while True:
    n = float(input())
    if n <= 0:
        break
    S += n
    k += 1
print('Среднее арифметическое положительных чисел равно %.3f' %(S/k))
```

6.5 Оператор continue

Оператор `continue` используется в случае, когда необходимо пропустить все оставшиеся команды в текущем блоке цикла и продолжить со следующей итерации цикла. Например:

```
S = k = 0
print('Введите числа в столбик (условие завершения: число 0)')
while True:
```

```

n = float(input())
if n < 0:
    continue
elif n == 0:
    break
S += n
k += 1
print('Среднее арифметическое положительных чисел равно %.3f' %(S/k))

```

6.6 Вопросы для текущего контроля успеваемости

- 1) Опишите структуру условного оператора (if). Приведите примеры записи его в сокращенном и полном форматах.
- 2) Приведите примеры циклических алгоритмов.
- 3) Какую структуру имеет оператор цикла for? В каких случаях целесообразно использование этого цикла? А в каких случаях его применение будет невозможно?
- 4) Приведите структуру и примеры использования цикла while.
- 5) Для чего используются операторы break и continue?

7 СТРУКТУРЫ ДАННЫХ

Структуры данных – это объекты, которые могут хранить некоторые последовательности данные вместе и восприниматься как единый объект. В Python существуют четыре встроенных структуры данных: список (list), кортеж (tuple), словарь (dict), множество (set) и строка (str).

7.1 Список

Список – это структура данных, которая хранит последовательность элементов в определенном порядке. Элементы списка разделяются запятыми и заключаются в квадратные скобки. Индексы (порядковые номера) элементов списка начинается с 0. Как только список создан, его можно изменять (редактировать): добавлять, удалять, искать или сортировать элементы.

В Python список может содержать данные разных типов (в отличие от статически типизированных языков программирования). Например:
list1 = ['строка', 'число', 2, 5, [1, 3], 1.5]

К его отдельным элементам можно обращаться по имени списка и индексу элемента: list1[0], list1[1].

7.1.1 Примеры создания списков

- 1) a = ['1', 2, 3.0]
- 2) a = list() # пустой список или
a = []
- 3) b = [0, 1]*5 # [0, 1, 0, 1, 0, 1, 0, 1, 0, 1]
- 4) list1 = [1, 2, 3, 4, 5]
list2 = ['один', 'два', 'три']
list3 = list1 + list2 # list3 как результат сложения двух списков

```
list4 = [list1, list2] # list4 – список, содержащий другие списки
5) a = input('Введите элементы через пробел: ').split()
    # элементы списка – строки
6) list('список') # ['с', 'п', 'и', 'с', 'о', 'к']
7) b = list(map(int, input('Введите числа через пробел: ').split()))
    # элементы списка – целые числа
8) c = list(zip(range(5), range(1,10,2))) # [(0, 1), (1, 3), (2, 5), (3, 7), (4, 9)]
```

Списки можно создавать с помощью генератора списков, применяя некоторое выражение (или функцию) к каждому элементу последовательности.

```
9) d = [c * 3 for c in 'list'] # ['lll', 'iii', 'sss', 'ttt']
10) list5 = [i * 2 for i in range(6)] # [0, 2, 4, 6, 8, 10]
11) list6 = [x for x in list5 if x > 5] # [6, 8, 10]
12) def plus(x, y):
    return x + y
    s = [plus(x, y) for x, y in zip(range(5), range(1,10,2))] # [1, 4, 7, 10, 13]
13) p = list(map(plus, range(5), range(1,10,2))) # [1, 4, 7, 10, 13]
```

Если некоторой переменной присвоить другую переменную, которая ссылается на список, то обе переменные будут связаны с одними тем же объектом (списком). Например:

```
list1 = [1, 2, 3, 4]
list2 = list1
list2[1] = 4
print(list1) # list1=[1, 4, 3, 4, 5]
```

Чтобы избежать этого, list1 надо не просто присвоить, а скопировать:

```
list2 = list1.copy()
```

7.1.2 Функции и методы для работы со списками

Оператор цикла for позволяет последовательно обращаться ко всем элементам списка:

```
for i in list1:
    print(i) # Элементы списка будут выведены в столбик
```

Функция len – определяет длину списка (количество элементов):

```
len(list1)
```

del используется для удаления элемента из списка или идентификатора объекта (в данном примере списка):

```
del list1[3]
del list1
```

```
len(list1) # NameError: name 'list1' is not defined
```

Основные методы для работы со списком приведены в таблице 5. Необязательные параметры заключены в квадратные скобки (они не входят в синтаксис языка Python).

Таблица 5

Метод списка	Назначение
<code>list.append(x)</code>	Добавляет элемент <code>x</code> в конец списка
<code>list.extend(L)</code>	Расширяет список <code>list</code> , добавляя в конец все элементы списка <code>L</code>
<code>list.insert(i, x)</code>	Вставляет на <code>i</code> -ое место в списке элемент со значением <code>x</code> , последующие элементы сдвигаются
<code>list.remove(x)</code>	Удаляет первый элемент в списке, имеющий значение <code>x</code>
<code>list.pop([i])</code>	Удаляет <code>i</code> -ый элемент и возвращает его. Если индекс не указан, удаляется последний элемент
<code>list.index(x, [start[, end]])</code>	Возвращает положение первого элемента от <code>start</code> до <code>end</code> со значением <code>x</code>
<code>list.count(x)</code>	Возвращает количество элементов со значением <code>x</code>
<code>list.sort([key = функция])</code>	Сортирует список на основе функции
<code>list.reverse()</code>	Разворачивает список – переписывает элементы в обратном порядке
<code>list.copy()</code>	Поверхностная копия списка
<code>list.clear()</code>	Очищает список

Обмен местами максимального элемента и элемента с нулевым индексом:

```
i = a.index(max(a))
a[0], a[i] = a[i], a[0]
```

Сортировка элементов массива в соответствии с заданным критерием:

```
a.sort(reverse=True) # сортировка массива a по убыванию
def sortByLength(inputStr):
    return len(inputStr)
a.sort(key = sortByLength) # сортировка массива a по ключу –
# возрастанию длины элементов
```

Списки позволяют выполнять действия над математическими объектами векторами (одномерными массивами) и матрицами (двумерными массивами).

7.1.3 Векторы

Ввод элементов целого типа (в столбик), когда количество элементов заранее известно:

```
n = int(input('n='))
print('Введите целые числа в столбик')
a = []
for i in range(n):
    a.append(int(input()))
```

Ввод элементов массива целого типа, когда количество элементов заранее неизвестно:

```
b = [int(i) for i in input('Введите элементы через пробел: ').split()]
```

Поиск и вывод элементов вектора в соответствии с заданным критерием:

```
for i in a:
```



```

if i < 0:
    print(i, end = ' ') # вывод отрицательных элементов
    Иначе отрицательные элементы вектора a можно вывести так:
print([i for i in a if i < 0])
    Вывод элементов вектора a до первого отрицательного:
for i in a:
    if i < 0:
        break
    print(i, end = ' ')
    Умножение вектора a на число k:
a = [k*a[i] for i in range(len(a))]
    Увеличение нечетных элементов вектора a в 2 раза:
k = len(a)
for i in range(k):
    if a[i] % 2 == 1:
        a[i] *= 2

```

7.1.4 Матрицы

Ввод элементов матрицы:

```

m = int(input('Введите количество строк '))
print('Введите матрицу ')
a = [[int(j) for j in input().split()] for i in range(m)]
    Проверка матрицы на корректность ввода (количество элементов во всех
строках должно быть одинаковое):
n = len(a[0])
k = 0
for i in range(1, m):
    if n == len(a[i]):
        k += 1
    else:
        break
if k == m-1:
    print('Матрица введена корректно')
else:
    print('Матрица введена неправильно')

```

Ввод элементов целочисленной матрицы, когда заранее известно число строк и столбцов в ней:

```

m = int(input('Введите количество строк '))
n = int(input('Введите количество столбцов '))
print('Введите матрицу ')
    # Все элементы вводятся в столбик,
    # сначала первая строка, затем вторая и т.д.
a = [[int(input()) for j in range(n)] for i in range(m)]
    Создание нулевой матрицы  $b(m \times n)$ :
b = [[0]*n for i in range(m)]

```

Вывод матрицы:

1) `print(a)` # вывод матрицы в виде списков в списке

2) `for x in a:` # *x* – это строка матрицы
`print(x)` # вывод матрицы построчно

3) `for x in a:`
`for j in x:`
`print('%3d' %j, end='')`
`print()` # вывод матрицы в виде таблицы без границ

4) `for i in range(m):`
`for j in range(n):`
`print('%3d' %a[i][j], end='')`
`print()` # вывод элементов матрицы по их индексам

Отбор и изменение элементов матрицы в соответствии с заданным критерием:

1) `S = 0`
`for j in range(n):`
`S+=a[2][j]` # Сумма элементов строки с индексом 2

2) `S = sum(a[2])` # Суммирование элементов строки можно выполнить проще

3) `S = 0`
`for x in a:`
`S+=x[0]` # Сумма элементов столбца с индексом 0

4) `S = sum([x[0] for x in a])` # Другой способ суммирования элементов столбца
Умножение матрицы *a* на число *k*:

1) `for x in a:`
`for j in range(len(x)):`
`x[j] *= k`

2) `a = [[j*k for j in x] for x in a]`
Умножение положительных элементов матрицы *a* на число *k*:

1) `for i in range(len(a)):`
`for j in range(len(a[i])):`
`if a[i][j] > 0:`
`a[i][j] *= k`

2) `def func(x, n):`
`if x > 0:`
`return x*n`
`return x`

`a = [[func(j, k) for j in x] for x in a]`

3) `a = [[(lambda x, n: x*n if x > 0 else x)(j, k) for j in x] for x in a]`

Пример 7.1. Умножить вектора *b*(*n*) на матрицу *a*(*n*×*p*).

$$\bar{c} = \bar{b} \cdot A = (b_1 \quad b_2 \quad \dots \quad b_n) \cdot \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1p} \\ a_{21} & a_{22} & \dots & a_{2p} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{np} \end{pmatrix} =$$

$$= (b_1 \cdot a_{11} + b_2 \cdot a_{21} + \dots + b_n \cdot a_{n1} \quad b_1 \cdot a_{12} + b_2 \cdot a_{22} + \dots + b_n \cdot a_{n2} \quad \dots \quad b_1 \cdot a_{1p} + b_2 \cdot a_{2p} + \dots + b_n \cdot a_{np})$$

1) c = []

for j in range(p):

element = 0

for i in range(n):

element += b[i]*a[i][j]

c.append(element)

2) c = [sum([(lambda x, n: x*n)(b[i], a[i][j]) for i in range(n)]) for j in range(p)]

Особое значение в математике уделяется квадратным матрицам, например, для них вводятся понятия главной и побочной диагоналей.

Вывод элементов главной диагонали матрицы a(n×n):

for i in range(n):

print(a[i][i], end=' ')

Транспонирование матрицы a(n×n):

for i in range(n):

for j in range(i):

a[i][j], a[j][i] = a[j][i], a[i][j]

Транспонирование матрицы a(m×n):

b = [[a[i][j] for i in range(m)] for j in range(n)]

Определитель матрицы с второго порядка:

det = c[0][0] * c[1][1] - c[0][1] * c[1][0]

Создание копии матрицы a:

1) d = []

for x in a:

d.append(x.copy())

2) d = [[j for j in x] for x in a]

3) from copy import deepcopy

d = deepcopy(a)

7.2 Кортеж

Кортежи служат для хранения нескольких объектов вместе. Их можно рассматривать как аналог списков, но с ограниченной функциональностью. Они неизменяемы.

Кортежи обозначаются указанием элементов, разделенных запятыми, которые заключаются в круглые скобки (скобки иногда можно опускать).

Например, print(1, 2, 3) выводит три числа, а print((1, 2, 3)) выводит кортеж. Обращение к элементам кортежа выполняется так же, как и к элементам списка. Функция len позволяет определять количество элементов в кортеже.

```
z = 1, 1, 2, 3
print(z[0])
```

Пустой кортеж создается при помощи пустой пары скобок: `t = ()` или `t = tuple()`. Кортеж из одного элемента нужно указывать при помощи запятой после первого (и единственного) элемента. Например:

```
a = 2,
a1 = (2,)
print((2,))
print(len(a1))
```

Можно создавать кортежи, включающие другие кортежи (аналогично двумерному массиву) или сочетать кортежи с другими структурами данных. Например:

```
t1 = [1, 4, 5], 2, 0, (1, 7)
print(t1) # ([1, 4, 5], 2, 0, (1, 7))
print(t1[0][2], t1[2], t1[3][0]) # 5 0 1
```

При попытке изменить элемент кортежа, например:

```
t1[2] = 10
```

Будет выведено сообщение об ошибке:

```
TypeError: 'tuple' object does not support item assignment
```

Однако изменить значение элемента списка, входящего в кортеж можно:

```
t1[0][0] = 3
print(t1) # ([3, 4, 5], 2, 0, (1, 7))
```

7.3 Словарь

Словарь в Python – это неупорядоченная структура произвольных объектов с доступом по ключу (ключ должен быть уникальным). Их иногда ещё называют ассоциативными массивами или хеш-таблицами (структура данных, которая позволяет хранить пары (ключ, значение) и выполнять три операции: добавления новой пары, поиск и удаление пары по ключу).

Основные свойства словарей:

- словари, так же как списки и кортежи, хранят ссылки на объекты, а не сами объекты;
- по аналогии со списком, в словаре можно получить доступ к элементам в цикле, но только по ключам, а не по индексам;
- элементы словаря хранятся в неотсортированном виде, ключи могут храниться не в том порядке, в котором они были добавлены;
- словарь может хранить в качестве значений объекты любого типа, тип ключа должен быть неизменяемым (строка, число или кортеж).

Создать пустой словарь можно следующим образом:

```
d1 = dict()
d2 = {}
```

Элементы словаря заключаются в фигурные скобки, пары ключ (key) – значение (value) в словаре разделяются двоеточиями и перечисляются через запятые:

```
d = {key1: value1, key2: value2}
```

Примеры создания словарей:

```
1) key1 = 'one'
   key2 = 'two'
   d = {key1: 25, key2: 100}
   print(d) # {'one': 25, 'two': 100}
```

Если изменить значение идентификатора key1, например:

```
key1 = 1
```

Элементы словаря не изменятся:

```
print(d) # {'one': 25, 'two': 100}
```

```
2) d1 = dict([(1, 1), (2, 4)]) # {1: 1, 2: 4}
```

```
   d2 = dict(((1, 7), (3, 0), (2, 4))) # {1: 7, 3: 0, 2: 4}
```

```
3) d3 = dict(zip(range(5), range(4, 10))) # {0: 4, 1: 5, 2: 6, 3: 7, 4: 8}
```

```
   d4 = dict(zip(range(10), range(4, 10))) # {0: 4, 1: 5, 2: 6, 3: 7, 4: 8, 5: 9}
```

```
4) d5 = dict.fromkeys([1, 2, 3]) # {1: None, 2: None, 3: None}
```

– создает словарь по списку ключей с пустыми значениями

```
   d6 = {}.fromkeys((2, 4, 6), 3) # {2: 3, 4: 3, 6: 3}
```

– все значения в словаре будут одинаковые

```
   d7 = {}.fromkeys('stroka', 2) # {'s': 2, 't': 2, 'r': 2, 'o': 2, 'k': 2, 'a': 2}
```

```
5) d = {a: a ** 2 for a in range(7)} # {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36}
```

– создание словаря с помощью генератора словарей

При создании словарей 4 и 5 способами можно использовать функции (по аналогии с созданием списков с помощью генератора).

Добавление пар ключ: значение в словарь выполняется так:

```
d[three] = 50 # {'one': 25, 'two': 100, 'three': 50}
```

Пусть задан словарь:

```
dict1 = {'ЭВМ': 'Электронно-вычислительная машина',
```

```
        'ПО': 'Программное обеспечение',
```

```
        'ООП': 'Объектно-ориентированное программирование'}
```

Добавление данных пользователя в существующий словарь:

```
dict1[input('Введите сокращение: ')] = input('Введите расшифровку: ')
```

Основные методы для работы со словарями приведены в таблице 6. Необязательные параметры заключены в квадратные скобки (они не входят в синтаксис языка Python).

Таблица 6

Метод словаря	Назначение
dict.clear()	Очищает словарь
dict.copy()	Возвращает копию словаря
dict.keys()	Возвращает ключи в словаре
dict.values()	Возвращает значения в словаре
dict.items()	Возвращает пары (ключ: значение)
dict.get(key[, default])	Возвращает значение для ключа key. Если ключ не найден, то возвращает default (по умолчанию None)

Метод словаря	Назначение
<code>dict.setdefault(key [, default])</code>	возвращает значение для ключа <code>key</code> . Если ключа нет, то создает ключ со значением <code>default</code> (по умолчанию <code>None</code>)
<code>dict.pop(key[, default])</code>	Удаляет ключ и возвращает значение. Если ключа нет, возвращает <code>default</code> (иначе вызывает исключение <code>KeyError</code>)
<code>dict.popitem()</code>	Удаляет и возвращает пару (ключ, значение). Если словарь пуст, вызывает исключение <code>KeyError</code>
<code>dict.update([other])</code>	Обновляет словарь, добавляя пары (ключ, значение) из <code>other</code> . Существующие ключи перезаписываются. Возвращает <code>None</code> (не новый словарь)

Количество терминов в словаре `dict1`:

```
len(dict1))
```

Вывод ключей словаря `dict1`:

```
for key in dict1:
```

```
    print(key)
```

Вывод значений словаря `dict1`:

```
for stat in dict1.values():
```

```
    print(stat)
```

Удаление данных из словаря `dict1` по ключу:

```
del dict1['ЭВМ']
```

Вывод элементов в виде ключ: значение:

```
1) for key, stat in dict1.items():
```

```
    print(key, ': ', stat)
```

```
2) for key in dict1:
```

```
    print(key, ': ', dict1.get(key))
```

Проверка наличия данных в словаре:

```
print('ПО' in dict1) # ключей
```

```
print('Программное обеспечение' in dict1.values()) # значений
```

Сортировка элементов в словаре:

```
sorted(d.keys()) # ключей
```

```
sorted(d.values()) # значений
```

```
for key in sorted(d.keys()):
```

```
    print(key, ': ', d[key]) # вывод отсортированных пар ключ : значение
```

Пример 7.2. Создать массив словарей, имеющих следующую структуру:

Наименование товара	Цена, р.	Количество
A(20)	9(8),9(2)	9(6)

а) Вывести данные о товаре по введенному наименованию.

б) Вычислить общую стоимость товаров.

```
stock = []
```

```
n = int(input('Введите количество товаров: '))
```

```

for i in range(n):
    print()
    product = {}
    product['name'] = input('Введите наименование товара: ')
    product['price'] = float(input('Введите цену (p.): '))
    product['number'] = int(input('Введите количество: '))
    stock.append(product)
print("\n Вывод данных по наименованию товара\n")
name = input('Введите интересующее наименование: ')
print("\nДанные о товаре: ',name)
fl = False
Summa = 0
for i in range(n):
    if stock[i].get('name') == name:
        print('Цена (p.): ',stock[i].get('price'))
        print('Количество: ',stock[i].get('number'))
        print()
        fl = True
        Summa+=stock[i]['price']*stock[i]['number']

```

```

if fl:
    print('Общая стоимость товаров (p.): ',Summa)
else:
    print('Данные о товарах отсутствуют')

```

Пример 7.3. Определить, сколько раз в строке встречается каждый символ.

```

def count_chars(s):
    d = dict()
    for c in s:
        if c not in d:
            d[c] = 1
        else:
            d[c] += 1
    return d
dict1 = count_chars(input('Введите текст:\n'))
for key in sorted(dict1):
    print(key, ':', dict1[key])

```

Пример 3. Инвертирование словаря (обмен ключей и значений).

```

def invert_dict(olddict):
    newdict = {}
    for key, value in olddict.items():
        newdict.setdefault(value, []).append(key)
    return newdict
d = {'child1': 'parent1', 'child2': 'parent1', 'child3': 'parent2', 'child4': 'parent2'}
print (invert_dict(d)) # {'parent1': ['child1', 'child2'], 'parent2': ['child3', 'child4']}

```

7.4 Множество

Множество – это неупорядоченный набор простых объектов. Оно используется тогда, когда важен только факт наличия объекта в наборе.

Примеры создания множеств:

- 1) `s1 = set()` # Пустое множество
- 2) `s2 = set('helloworld')` # {'l', 'e', 'o', 'w', 'h', 'd', 'r'}
- 3) `s3 = {i ** 2 for i in range(10)}` # генератор множества
{0, 1, 4, 81, 64, 9, 16, 49, 25, 36}

Для множеств (например, `s2` и `s3`) можно выполнять:

- определение длины – количества объектов во множестве (`len(s2)`),
- добавление элементов (`s2.add(1)`),
- проверку принадлежности объекта (`a in s2`), равенства множеств (`s2==s3`),
- операции: объединение (`s2 | s3`), пересечение (`s2 & s3`), разность (`s3 - s2`), исключающее или (`s3 ^ s2`).

Остальные методы для работы с множествами можно прочитать в справке для множества: `help(set)`.

Множество `set` – это изменяемый тип данных. Если есть необходимость оставить все элементы множества без изменений, то следует использовать тип `frozenset`.

7.5 Строка

Строки, как константы, описаны в п. 3.5. Это упорядоченные последовательности символов, каждый из которых имеет порядковый номер (индекс), нумерация начинается с 0. К отдельным элементам строк (символам) можно обращаться по их индексам. Например, `s[1]`.

Длина строки (количество символов) определяется с помощью функции `len()`. Для строк можно использовать операции сложения (конкатенации): `s1 + s2`, дублирования строки `s1 * n` (где `n` – целое число).

Некоторые методы для работы со строками содержатся в таблице 7.

Таблица 7

Метод строки	Назначение
<code>str.find(str, [start[, end]])</code>	Поиск подстроки в строке. Возвращает номер первого вхождения или -1
<code>str.rfind(str, [start[, end]])</code>	Поиск подстроки в строке. Возвращает номер последнего вхождения или -1
<code>str.split(символ)</code>	Разбиение строки по разделителю
<code>str.isdigit()</code>	Проверка, состоит ли строка из цифр
<code>str.isalpha()</code>	Проверка, состоит ли строка из букв
<code>str.isalnum()</code>	Проверка, состоит ли строка из цифр или букв
<code>str.islower()</code>	Проверка, состоит ли строка из символов в нижнем регистре
<code>str.isupper()</code>	Проверка, состоит ли строка из символов в верхнем регистре

Метод строки	Назначение
<code>str.istitle()</code>	Проверка, начинаются ли слова в строке с заглавной буквы
<code>str.upper()</code>	Возвращает строку, преобразованную к верхнему регистру
<code>str.lower()</code>	Возвращает строку, преобразованную к нижнему регистру
<code>S.startswith(stroka)</code>	Проверка, начинается ли строка S с шаблона stroka
<code>S.endswith(stroka)</code>	Проверка, заканчивается ли строка S шаблоном stroka
<code>S.join(список)</code>	Сборка строки из списка с разделителем S
<code>str.capitalize()</code>	Возвращает строку, в которой первый символ переведен в верхний регистр, а все остальные – в нижний
<code>str.lstrip()</code>	Возвращает строку, в которой удалены пробельные и экранированные символы в начале строки
<code>str.rstrip()</code>	Возвращает строку, в которой удалены пробельные и экранированные символы в конце строки
<code>str.strip()</code>	Возвращает строку, в которой удалены пробельные и экранированные символы в начале и в конце строки
<code>str.title()</code>	Возвращает строку, в которой первая буква каждого слова переведена в верхний регистр, а все остальные – в нижний

Примеры соединения элементов списка и словаря в строки:

```
a = ['H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd', '!']
''.join(a) # 'Hello World!'
'_'.join(a) # 'H_e_l_l_o_ _W_o_r_l_d_!'
d = {'1':'one', '2':'two', '3':'three'}
'.'.join(d) # '1-2-3'
'.'.join(d.values()) # 'one_two_three'
```

7.6 Срезы последовательности (Slicing)

Срезы позволяют получать новые последовательности из других упорядоченных последовательностей (списков, кортежей, строк). Структура среза: `obj[start : end : step]`, где `obj` – идентификатор объекта-последовательности, `start` – начальный индекс, `end` – конечный индекс, `step` – шаг (если они отсутствуют, то принимают значения по умолчанию). Индексы и шаг могут принимать отрицательные значения (в этом случае отсчет начинается от последнего символа, индекс которого -1).

Примеры срезов строки:

```
st = 'Язык программирования Python'
st[:4] # 'Язык'
st[5:] # 'программирования Python'
st[-6:] # 'Python'
```

```
st[1:-1] # 'зык программирования Pytho'  
st[-1:-7:-1] # 'nohtyP'  
st[:] # 'Язык программирования Python'  
st[::-1] # 'nohtyP яинавориммаргорп кызЯ'  
st[::3] # 'Якppмон tn'
```

Примеры срезов списка:

```
a = [2*i for i in range(10)] # [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]  
a[: :-1] # [18, 16, 14, 12, 10, 8, 6, 4, 2, 0]  
c = a  
c is a # True  
b = a[:] # так можно создать копию списка  
b is a # False  
a[2: 4] = [0, 0] # a == c == [0, 2, 0, 0, 8, 10, 12, 14, 16, 18]
```

7.7 Вопросы для текущего контроля успеваемости

- 1) Что такое структура данных? Ее основные отличия от данных простого типа.
- 2) Приведите примеры списков, обращения к элементам списка, обработки данных с помощью методов списка.
- 3) Что общего и различного между списками и кортежами? Приведите примеры.
- 4) Какая структура данных называется словарем? Приведите пример словаря и применения к нему методов.
- 5) Что такое множества? В каких случаях их целесообразно использовать?
- 6) Приведите примеры строк и методов их обработки.
- 7) С какой целью используются срезы? Приведите примеры.

8 ФАЙЛЫ

Файл – это именованный участок долговременной (дисковой) памяти. В зависимости от способа обработки данных файлы делятся на текстовые и бинарные (или двоичные – последовательность байтов).

Язык программирования Python предоставляет различные средства для работы с каждым типом файлов. Основные операции, выполняемые над файлами: открытие, закрытие, чтение, запись, дозапись, переход на нужный символ или байт (прямой доступ к данным). В конце файла содержится признак его завершения.

Метод открытия файла возвращает файловый объект, имеет один обязательный параметр (имя файла – file_name), значения остальных параметров задаются по умолчанию, их можно изменять:

```
open(file_name [, mode = 'rt', buffering = -1, encoding = None, errors = None,  
newline = None, closed = True, opener = None])
```

Путь к файлу может быть относительным или абсолютным. Например:
name = 'text1.txt' (файл содержится в текущей папке);

name1 = r'd:\files\text1.txt' (указан путь доступа к файлу) – «сырая» строка, иначе можно при записи полного имени файла использовать экранирование:
name2 = 'd:\\files\\text2.txt'

Второй аргумент (mode) – это режим, в котором открывается файл.

Таблица 8

Режимы (mode) метода open

Режим	Обозначение
'r'	на чтение (является значением по умолчанию)
'w'	на запись, содержимое файла удаляется, если файла не существует, создается новый
'x'	на запись, если файла не существует, иначе исключение
'a'	на дозапись, информация добавляется в конец файла
'b'	в двоичном режиме
't'	в текстовом режиме (является значением по умолчанию)
'+'	на чтение и запись

Режимы могут быть объединены, например, 'rb' – чтение в двоичном режиме. Режим '+' может быть добавлен к остальным режимам. По умолчанию режим открытия равен 'rt'.

При открытии файла необходимо выполнять проверку на возможность выполнения этой операции.

try:

```
f=open(name,'x')
except FileExistsError:
    print('File exist')
```

try:

```
f=open(name,'w')
except PermissionError:
    print('File is Permission')
```

try:

```
f=open(name,'r')
except FileNotFoundError:
    print('File not exist')
```

Третий параметр устанавливает размер буферизации при работе с файлом. По умолчанию он выключен, и чтение/запись идет напрямую с диска на диск. Для включения буфера третий параметр должен быть отличным от нуля.

Варианты буферизации (buffering): 0 – отключить буферизацию (только для двоичного режима); 1 – построчная буферизация (только для текстового режима); > 1 – размер буфера в байтах.

Аргумент encoding задает кодировку, используется только в текстовом режиме чтения файла. Например,
f = open(name, 'r', encoding = 'utf-8')

Метод закрытия файла выполняет перенос данных из буфера в файл (при необходимости), затем закрывает файл (доступ к данным из программы становится невозможным). Например,
f.close()

8.1 Текстовые файлы

Основное преимущество текстовых файлов заключается в простоте работы с ними (по сравнению с бинарными файлами), а также в возможности их создания, открытия на чтение или изменение в текстовом редакторе. Все данные, хранящиеся в файле, воспринимаются как текст, разделенный на строки, состоящие из символов и заканчивающиеся символом конца строки – '\n'.

1) Запись данных в файл осуществляется с помощью метода write(), который возвращает число записанных символов. Например,

```
f = open('text.txt', 'w') # Открыть файл на запись
d = ['1716', '1817', '1918']
for x in d:
    f.write(x + '\n')
```

Запись нескольких строк в файл выполняется с помощью метода writelines(). Например,

```
f.writelines(d) # В файл будет записана одна строка '171618171918'
d = [x + '\n' for x in d] # Добавление символа конца строки к каждому элементу списка
f.writelines(d) # В файл будут записаны три строки
```

2) Чтение данных из файла.

Метод read() читает весь файл целиком, если был вызван без аргументов. Если задать read(n), где n – целое число, то будет считано n символов. Например,

```
s = f.read(1) # Считывается один символ: s = '1'
s = f.read() # Считываются все остальные символы до конца файла: s = '171618171918'
```

Для того чтобы прочитать файл *построчно*, можно воспользоваться циклом for. Например,
for line in f:

```
    print(line.strip()) # strip() удаляет символы конца строки
```

Можно создать список из строк файла: d = [line.strip() for line in f]

Чтение одной строки из файла выполняется с помощью метода f.readline(), который без параметра читает всю строку. Если задать параметр, то он задает максимальное число символов строки, которое будет прочитано. Например, readline(3) прочитает 3 символа строки, при повторном вызове readline(6) прочитает еще 6 символов, либо столько символов, сколько их осталось до конца строки. В отличие от нее read(6) может при чтении брать символы из разных строк. При этом '\n' воспринимается как один символ.

Для чтения в список всех строк из файла предназначен метод: `f.readlines()`.

Существует оператор `with ... as ...`, который позволяет выполнять последовательность операций: открытие файла, выполнение действий над его данными, закрытие файла. Например,

```
with open(name) as f:
```

```
    d = f.readlines()
```

```
with open(name) as f:
```

```
    for line in f:
```

```
        print(line.rstrip())
```

Пример 8.1. Дан текстовый файл. Прочитать из него все строки и записать каждую строку в обратном порядке следования символов в другой файл.

```
name1 = r'd:\files\text1.txt'
```

```
name2 = 'd:\\files\\text2.txt'
```

```
with open(name1) as f1, open(name2,'w') as f2:
```

```
    for line in f1:
```

```
        f2.write(line[::-1].strip()+'\n')
```

3) Произвольный доступ к данным файла.

По умолчанию метод `read()` читает данные последовательно по порядку, от начала и до конца файла. Для произвольного доступа к файлу используется метод: `seek(offset[, whence])`, где `offset` – смещение в байтах относительно начала файла; `whence` – указывает на то, от чего отсчитывается смещение: 0 – от начала файла (по умолчанию); 1 – от текущей позиции; 2 – от конца файла.

Если файл открыт в режиме добавления данных (а или а+) любые изменения, сделанные функцией `seek()` будут отменены при последующей записи. Если файл открыт в текстовом режиме, допускается указание только смещений, поведение при указании других значений не определено.

Метод `f.tell()` возвращает текущую позицию файла.

Пример 8.2. Считать данные из файла и вывести список всех товаров на терминал и в файл.

```
# Преобразование строки в словарь
```

```
def line_in_dict(st):
```

```
    d = {}
```

```
    prod = st.strip().split()
```

```
    d['name'] = prod[0]
```

```
    d['price'] = float(prod[1])
```

```
    d['number'] = int(prod[2])
```

```
    return d
```

```
# Вывод списка товаров
```

```
def print_tovar():
```

```
    f1 = open(name1)
```

```
    print('\n
```

```
        V002')
```

```
    print('        Список товаров на складе')
```

```
    print('
```

```
    print(' | Наименование товара | Цена, р. | Количество, шт. |')
```

```

print(' |-----|-----|-----| ')

for line in f1:
    product = line_in_dict(line)
    print(' | %-20s | %8.2f | %4i |'%(product['name'],
product['price'], product['number']))
    print(' |-----|-----|-----| ')
f1.close()

# Вывод списка товаров в файл
def printfile_tovar():
    f1 = open(name1)
    f2 = open(name2, 'w', encoding='utf-8')
    f2.write('\n                                     D002\n')
    f2.write('          Список товаров на складе\n')

f2.write(' |-----|-----|-----| \n')
f2.write(' | Наименование товара | Цена, р. | Количество,
шт. | \n')

f2.write(' |-----|-----|-----| \n')

for line in f1:
    product = line_in_dict(line)
    f2.write(' | %-20s | %8.2f | %4i
|\n'%(product['name'], product['price'], product['number']))

f2.write(' |-----|-----|-----| \n')
f1.close()
f2.close()

name1 = r'd:\files\text1.txt'
name2 = r'd:\files\text2.txt'
print_tovar()
printfile_tovar()

```

Результаты работы

Содержимое файла r'd:\files\text1.txt'

```

ручка          15.00  5
карандаш      8.10   7
линейка       19.50  6

```

V002

Список товаров на складе

Наименование товара	Цена, р.	Количество, шт.
ручка	15.00	5
карандаш	8.10	7
линейка	19.50	6

Список товаров на складе

Наименование товара	Цена, р.	Количество, шт.
ручка	15.00	5
карандаш	8.10	7
линейка	19.50	6

8.2 Бинарные файлы

Двоичный (бинарный) файл хранит последовательность байтов. В нем нет разделения по типам данных. При записи данных в файл их надо преобразовать в последовательность байт (сделать это можно разными способами), при чтении данных из файла их надо правильно интерпретировать.

При открытии бинарного файла на чтение или запись также надо учитывать, что нам нужно применять режим "b" в дополнение к режиму записи ("w") или чтения ("r").

- 1) Модуль pickle позволяет записывать объекты в файл и считывать их.

`import pickle` – импорт модуля в программу;

`dump(obj, file)` – запись объекта `obj` в бинарный файл `file`;

`obj = load(file)` – чтение данных из бинарного файла в объект `obj`.

Структура объектов должна быть строго определена, иначе возникнут ошибки при чтении данных.

Пример 8.3. Записать данные из словаря в двоичный файл, затем считать их из файла.

```
import pickle
```

```
# Добавление данных в словарь
```

```
def add_tovar():
```

```
    product = { }
```

```
    product['name'] = input('Введите наименование товара: ')
```

```
    product['price'] = float(input('Введите цену: '))
```

```
    product['number'] = int(input('Введите количество: '))
```

```
    return product
```

```
f = open('sklad.dat', 'wb')
```

```
pickle.dump(add_tovar(), f)
```

```
f = open('shoplistfile', 'rb')
```

```
fl = True
```

```
while fl:
```

```
    try:
```

```
        product = pickle.load(f)
```

```
        print(product)
```

```
        print(f.tell())
```

```
    except EOFError:
```

```
        print('error')
```

```
fl = False
```

2) Стандартные методы работы с двоичным файлом

Пример 8.4. Запись-чтение массива целочисленных данных размером 1 байт в беззнаковом формате.

```
a = [123, 3, 255, 0, 100]
```

```
b = bytes(a) # байтовый массив b'\x03\xff\x00d'
```

```
    # можно использовать метод b = bytearray(a)
```

```
    # результат которого bytearray(b'\x03\xff\x00d')
```

```
f = open('path.dat', 'w+b')
```

```
f.write(b)
```

```
f.seek(0) # Переход в начало файла
```

```
b1 = list(f.read()) # b1 = [123, 3, 255, 0, 100]
```

```
n = f.seek(0,2) # Переход в конец файла, а также определение длины файла
```

```
f.close()
```

Если надо записать и прочитать данные большего размера, то они сначала разбиваются на байты. Существуют два формата записи целых чисел размером больше одного байта:

от старшего байта к младшему (byteorder = 'big'): [123, 3] → 123*256+3 = 31491;

от младшего байта к старшему (byteorder = 'little'): [123, 3] → 3*256+123 = 891.

```
k = int.from_bytes([123, 3], byteorder = 'big')
```

Пример 8.5. Записать в файл массив из целочисленных данных размером 2 байта в беззнаковом формате:

```
a = [567, 764, 92, 1000]
```

```
d = []
```

```
for x in a:
```

```
    d.append(x // 256)
```

```
    d.append(x % 256) # d = [2, 55, 2, 252, 0, 92, 3, 232]
```

```
byte_d = bytes(d) # создание байтового массива
```

```
f.write(byte_d)
```

```
n = f.seek(0,2) # n = 8 – проверка правильности записи данных
```

```
f.seek(0)
```

```
g = [int.from_bytes(f.read(2), byteorder='big') for x in range(n // 2)]
```

```
    # g = [567, 764, 92, 1000]
```

Другой способ чтения данных из файла с использованием цикла:

```
f.seek(0)
```

```
c = []
```

```
while True:
```

```
    x=f.read(2)
```

```
    if x:
```

```
        c.append(int.from_bytes(x, byteorder = 'big'))
```

```
    else:
```

```
        break
```

```
    # c = [567, 764, 92, 1000]
```


Пример 8.6. Записать в файл строку символов и считать эту строку из файла.

```
s = bytes('stroka', 'utf-8') # b = b'stroka'
f = open('path.dat', 'w+b')
f.write(s)
f.flush() # очистка буфера записи
f.seek(0)
c = list(f.read()) # c = [115, 116, 114, 111, 107, 97] – коды символов
s = [chr(i) for i in c] # s = ['s', 't', 'r', 'o', 'k', 'a']
st = ''.join(s)
print(st) # st = 'stroka'
# Иначе:
f.seek(0)
b = f.read() # b = b'stroka'
st = str(b)[2:-1]
print(st) # st = 'stroka'
```

Если двоичный файл содержит только символы, то к нему можно обращаться бинарным (b) и в текстовом (t) режимах.

3) Модуль struct работает с упакованными двоичными данными.

Данный модуль позволяет записывать и считывать данные, используя формат, который указывает на тип данных и на размер (при работе со структурными объектами). Основные функции:

- struct.pack(format, x1, x2, ...) – возвращает строку, содержащую значения, упакованные в соответствии с заданным форматом (format). Аргументы x1, x2, ... должны точно соответствовать значениям, требуемым форматом;
- struct.unpack(format, x1, x2, ...) – распаковывает строку в соответствии с заданным форматом. Результатом является кортеж;
- struct.calcsize(format) – возвращает размер структуры, соответствующей данному формату.

Прочитать информацию обо всех функциях модуля struct можно по ссылке <https://docs.python.org/2/library/struct.html>

Пример 8.7. Выполнить задание из примера 8.3 с использованием модуля struct.

```
import struct
name = bytes(d['name']+'', 'utf-8')
data = struct.pack('20sfi', name, d['price'], d['number'])
len(data)
name, d['price'], d['number'] = struct.unpack('20sfi', data)
s = [chr(i) for i in name]
d['name'] = ''.join(s).strip()
# Иначе:
# d['name'] = str(name)[2:-1].strip()
```

8.3 Вопросы для текущего контроля успеваемости

- 1) На какие группы можно разделить файлы в зависимости от способа обработки?
- 2) Какие действия можно выполнять над файлами?
- 3) Перечислите режимы открытия файлов.
- 4) Какие методы используются чтения данных из текстового файла и записи в текстовый файл?
- 5) Какие модули и методы используются чтения данных из бинарного файла и записи в бинарный файл?

9 ГРАФИЧЕСКИЕ ВОЗМОЖНОСТИ МОДУЛЯ TKINTER

Tkinter (от англ. tk interface) – это графическая библиотека, позволяющая создавать программы с оконным интерфейсом. Эта библиотека является интерфейсом к языку программирования и инструменту создания графических приложений.

9.1 Общие сведения о Tkinter

Tkinter – это пакет для Python, предназначенный для работы с библиотекой Tk. Библиотека Tk содержит компоненты графического интерфейса пользователя (graphical user interface – GUI), написанные на языке программирования Tcl.

Под графическим интерфейсом пользователя (GUI) подразумеваются все окна, кнопки, текстовые поля для ввода, списки, радиокнопки, флажки и др., которые появляются на экране при запуске приложения. Через них пользователь взаимодействует с программой и управляет ее работой. Все эти элементы интерфейса вместе называются виджетами (widgets, от англ. window gadget).

В настоящее время почти все приложения, которые создаются для конечного пользователя, имеют GUI. Редкие программы, подразумевающие взаимодействие с человеком, остаются консольными.

Существует множество библиотек GUI. Однако библиотека Tk была выбрана для Python по умолчанию. Установочный файл Питона уже включает пакет tkinter в составе стандартной библиотеки наряду с другими модулями. Tkinter можно охарактеризовать как переводчик с языка Python на язык Tcl.

Приложения с графическим интерфейсом пользователя событийно-ориентированные. Событийно-ориентированное программирование базируется на объектно-ориентированном и структурном.

События бывают разными: временной фактор, нажатие кнопки мыши или клавиши Enter, ввод текст, прокрутка страницы вниз и т.д. В этих случаях создаются соответствующие обработчики событий, происходит срабатывание определенной части программы, что приводит к какому-либо результату.

Tkinter импортируется стандартно любым из способов: `import tkinter`, `from tkinter import *`, `import tkinter as tk`. Можно импортировать отдельные классы, что делается редко.

Далее, чтобы написать GUI-программу, надо выполнить следующий порядок действий:

- 1) Создать главное окно.
- 2) Создать виджеты и выполнить конфигурацию их свойств (опций).
- 3) Определить события (то есть то, на что будет реагировать программа).
- 4) Определить обработчики событий (то есть то, как будет реагировать программа).
- 5) Расположить виджеты в главном окне.
- 6) Запустить цикл обработки событий.

Модуль `tkinter` включает следующие классы:

- 1) `Button` (кнопка)
- 2) `Radiobutton` (радио-кнопка)
- 3) `Checkbutton` (флажок)
- 4) `Entry` (однострочное поле для ввода)
- 5) `Text` (многострочное поле для ввода)
- 6) `Label` (метка)
- 7) `Scale` (ползунок)
- 8) `Scrollbar` (полоса прокрутки)
- 9) `Frame` (виджет для группировки других виджетов)
- 10) `LabelFrame` (аналог `Frame`, только с заголовком)
- 11) `Listbox` (список)
- 12) `Canvas` (поле для рисования)
- 13) `PanedWindow` (элемент разделения окна)
- 14) `Menu` (главное меню)
- 15) `Tk` (главное единственное окно)
- 16) `Toplevel` (дочернее окно)

Прочитать информацию о классах модуля `tkinter` и их методах можно по ссылке https://ru.wikibooks.org/wiki/GUI_Help/Tkinter_book

Класс `Tk` является базовым классом любого `Tkinter` приложения. При создании объекта этого класса запускается интерпретатор `tcl/tk` и создаётся базовое окно приложения. Переменную, связываемую с объектом, часто называют `root` (корень).

`Tkinter` является событийно-ориентированной библиотекой. В приложениях такого типа имеется главный цикл обработки событий. В `Tkinter` такой цикл запускается методом `mainloop`. Для явного выхода из интерпретатора и завершения цикла обработки событий используется метод `quit`.

```
import tkinter as tk
import tkinter.colorchooser
root = tk.Tk()
root.title('Главное окно')
root.geometry('800x600+100+100')
c = tk.Canvas(root, width=700, height=500, bg='#8fff8f', bd=1)
c.pack(side=tk.RIGHT) # c.place(x=0,y=0)
b = tk.Button(root, bg='#8080ff', bd=2, text='Button', cursor='circle')
```

```

        # bitmap='warning',
b.pack()
def change(side=tk.LEFT):
    color = tk.colorchooser.askcolor()
    b['text'] = "Изменено"
    b['bg'] = c['bg'] = color[1]
    b['activebackground'] = '#555555'
    b['fg'] = '#ffffff'
    b['activeforeground'] = '#ffffff'
b.config(command = change)
root.mainloop()

```

#mainloop – обработка событий, пока окно не будет закрыто.

Свойства bg, fg, activebackground и activeforeground определяют соответственно цвет фона и текста, цвет фона и текста во время нажатия и установки курсора мыши над кнопкой.

9.2 Графические объекты пакета Tkinter

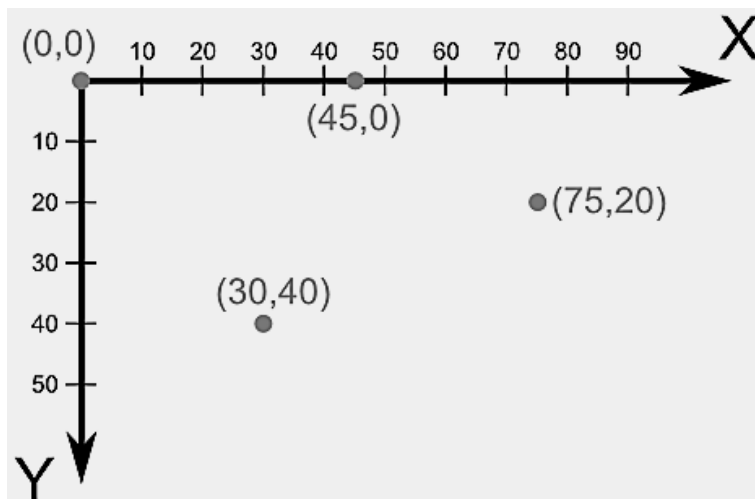
В Tkinter существует три менеджера геометрии – упаковщик, сетка и размещение по координатам. Упаковщик (packer) вызывается методом pack(), который имеется у всех виджетов-объектов. Если к элементу интерфейса не применить какой-либо из менеджеров геометрии, то он не отобразится в окне. При этом в одном окне (или любом другом родительском виджете) нельзя комбинировать разные менеджеры. Если стали размещать виджеты методом pack(), то не следует использовать методы grid() (сетка) и place() (место).

Если упаковщику не передавать аргументы, то виджеты будут располагаться вертикально, друг над другом. Тот объект, который первым вызовет pack(), будет сверху, второй объект – под первым, и так далее.

У метода pack() есть параметр side (сторона), который принимает одно из четырех значений-констант – TOP, BOTTOM, LEFT, RIGHT (верх, низ, лево, право). По умолчанию side = TOP.

В tkinter от класса Canvas создаются объекты-холсты, на которых можно "рисовать", размещая различные фигуры и объекты. Делается это с помощью вызовов соответствующих методов.

При создании экземпляра Canvas необходимо указать его ширину и высоту. При размещении геометрических примитивов и других объектов указываются их координаты на холсте. Точкой отсчета является верхний левый угол холста.



Например, построение линии и прямоугольника:

```
c.create_line(100, 180, 100, 60, fill = 'green', width = 5,
             arrow = tk.LAST, dash = (10,2),
             activefill = 'lightgreen',
             arrowshape = "10 20 10")
c.create_rectangle(160, 180, 240, 290, fill='yellow', outline='green',
                  width=3, activedash=(5, 4))
```

Сначала указываются координаты начала (x1, y1), затем – конца (x2, y2). Остальные свойства являются необязательными, например, activefill определяет цвет отрезка при наведении на него курсора мыши.

Первые координаты – верхний левый угол, вторые – правый нижний. В приведенном примере, когда на прямоугольник попадает курсор мыши, его рамка становится пунктирной, что определяется свойством activedash.

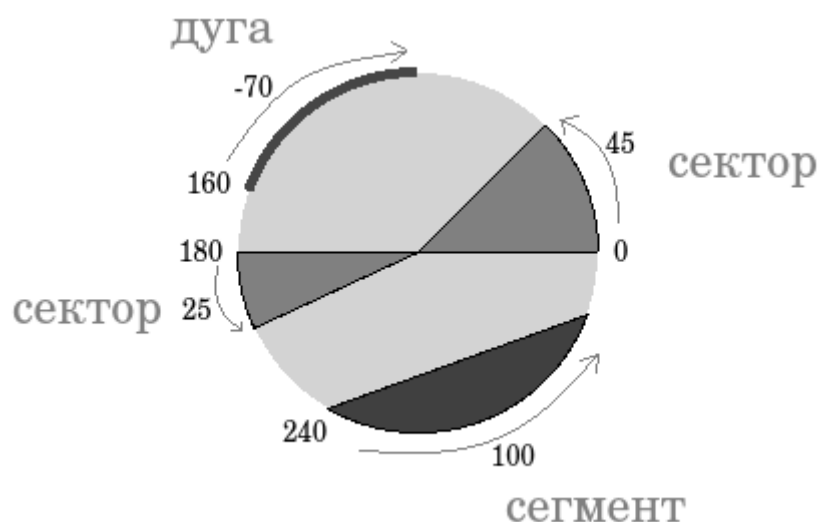
Методом create_polygon() рисуется произвольный многоугольник путем задания координат каждой его точки:

```
c.create_polygon(100, 10, 20, 90, 180, 90)
c.create_polygon(40, 110, 160, 110, 190, 180, 10, 180, fill = 'orange',
                outline = 'black')
```

Метод create_oval() создает эллипсы. При этом задаются координаты гипотетического прямоугольника, описывающего эллипс. Если нужно получить круг, то соответственно описываемый прямоугольник должен быть квадратом. Например,

```
c.create_oval(50, 10, 150, 110, width = 2)
c.create_oval(10, 10, 190, 190, fill = 'lightgrey', outline = 'white')
```

Построить часть круга (дугу, сектор или сегмент) можно с помощью функции arc. Правила указания начала отсчета и вычисления соответствующих центральных углов выполняются также, как на координатной плоскости (от положительного направления оси Oх против часовой стрелки). Примеры приведены во фрагменте программного кода.



```

c.create_arc(10, 10, 190, 190, start = 0, extent = 45, fill = 'red')
c.create_arc(10, 10, 190, 190, start = 180, extent=25, fill = 'orange')
c.create_arc(10, 10, 190, 190, start = 240, extent = 100, style = tk.CHORD, fill =
'green')
c.create_arc(10, 10, 190, 190, start = 160, extent = -70, style = tk.ARC, outline =
'darkblue', width = 5)

```

Вывод текста на холст выполняется с помощью функции `create_text`. Например,

```

c.create_text(100, 100, text = "Hello World,\nPython\nand Tk!",
justify = tk.CENTER, font = "Verdana 14")
c.create_text(200, 200, text = "About this",
anchor = tk.SE, fill = "grey")

```

По умолчанию в заданной координате располагается центр текстовой надписи. Чтобы изменить это и, например, разместить по указанной координате левую границу текста, используется якорь со значением `W` (от англ. *west* – запад). Другие значения: `N`, `NE`, `E`, `SE`, `S`, `SW`, `W`, `NW`. Если букв, задающих сторону привязки, две, то вторая определяет вертикальную привязку (вверх или вниз «уйдет» текст от заданной координаты). Свойство `justify` определяет лишь выравнивание текста относительно самого себя (`tk.CENTER`, `tk.LEFT`, `tk.RIGHT`).

Каждый объект имеет идентификатор, являющийся уникальными в пределах программы. При обращении к объекту можно использовать идентификатор, присвоенный ему автоматически, или заданный явно, например, `line = c.create_line(10, 10, 400, 200)`, где `line` – это идентификатор объекта. С их помощью можно изменять свойства объекта (координаты, цвет, стиль и т.д.).

Теги (`tag`) используются для того чтобы объединять объекты в группы и изменять в программе их свойства одновременно. Имя тега заключается в кавычки (строковое значение). Например,

```

oval = c.create_oval(30, 10, 130, 80, tag = "group1")
line = c.create_line(10, 100, 450, 100, tag = "group1")
c.itemconfig('group1', fill = "red", width = 3)

```

Эллипс и линия содержат один и тот же тег `group1`, а функция `color` изменяет цвет всех объектов с данным тегом.

На холст можно вывести картинки из файлов с расширениями `.gif`, `.pgm`, или `.ppm format`. Например,

```
P = tk.PhotoImage(file = 'фото1.gif')
p = c.create_image(10, 10, anchor = tk.NW, image = P)
```

9.3 Пример движение объектов на холсте

```
ball = c.create_oval(10,200,30,220,fill='#00ff00')
import time
for i in range(100):
    c.move(ball,2,2)
    form3.update()
    time.sleep(0.05)
c.coords(ball,10,200,30,220)
```

10 ОСНОВЫ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ (ООП). КЛАССЫ И ОБЪЕКТЫ

Python соответствует принципам объектно-ориентированного программирования. В Python всё является объектами: целочисленные данные, строки, списки, словари и всё остальное.

Но возможности ООП в Python этим не ограничены. Программист может написать свой тип данных (класс), определить в нём свои методы.

10.1 Основные понятия ООП

Абстрагирование – это описание характеристик некоторого объекта, которые четко определяют его назначение в рамках рассматриваемой программы и отличают его от всех других объектов.

Класс – тип, описывающий устройство объектов. Это ключевое понятие в объектно-ориентированном программировании. Класс содержит в себе данные и код, который управляет этими данными.

Имена классов принято начинать с прописной буквы.

```
class F():
```

```
    pass # Пример описания заготовки класса, который ничего не делает
```

Наследование – один из основных механизмов, позволяющий описать новый класс на основе уже существующего (родительского), при этом свойства и функциональность родительского класса заимствуются новым классом.

Классы-родители перечисляются после имени класса в скобках через запятую:

```
class SomeClass(ParentClass1, ParentClass2, ...):
```

```
    # атрибуты (поля) и методы класса SomeClass
```

Свойства атрибутов класса устанавливаются с помощью простого присваивания. Например:

```
class A(object):
```

```
x = 1
```

Методы класса объявляются как функции:

```
class SomeClass(object):  
    def method1(self, x):
```

```
        # код метода
```

Первый аргумент – `self` – общепринятое имя для ссылки на объект, в контексте которого вызывается метод. Этот параметр обязателен и отличает метод класса от обычной функции.

Конструктор класса – это специальный метод (`__init__`), который вызывается при создании нового объекта и используется для инициализации полей класса значениями, а также для начальных вычислений, если они необходимы. После создания объекта конструктор вызвать нельзя. Кроме того, данный метод никогда не возвращает никакого значения.

```
class B(object):  
    def __init__(self, x=0, y=0):  
        self.r = x  
        self.i = y
```

Классы в Python могут динамически изменяться после определения:

```
class SomeClass(object):  
    pass  
def squareMethod(self, x):  
    return x*x  
SomeClass.square = squareMethod  
obj = SomeClass()  
obj.square(5)
```

Экземпляр класса (Instance – инстанс) задается простым присваиванием:

```
a = A()  
b = A() # Создали два экземпляра a и b класса A()  
a.x = 5 # У экземпляра a появился собственный атрибут  
print(a.x) # 5  
print(b.x) # 1  
del(a.x) # Удаление атрибута у экземпляра  
print(a.x) # 1
```

При создании объекта-экземпляра выполняется следующая последовательность действий:

- создается новый экземпляр заданного класса и все его атрибуты инициализируются значениями по умолчанию;
- при создании объекта выполняется соответствующий конструктор с указанным списком параметров;
- формируется ссылка на созданный и инициализированный объект.

Объект класса может имитировать стандартную функцию, то есть при желании его можно «вызвать» с параметрами. За эту возможность отвечает специальный метод `__call__`:

```
class Multiplier:  
    def __call__(self, x, y):
```



```
return x*y
multiply = Multiplier()
multiply(19, 19) # 361
# то же самое
multiply.__call__(19, 19) # 361
```

Полиморфизм (буквально означает наличие нескольких форм) – в контексте ООП означает способность объекта вести себя по-разному.

Полиморфизм в программировании реализуется через перегрузку метода, либо через его переопределение.

При наследовании метод подкласса переопределяет метод класса-родителя только в случае, когда эти методы имеют одинаковые имена и списки параметров. Иначе выполняется перегрузка метода – свойство, позволяющее выбирать выполняемый метод, в зависимости от количества или типа параметров.

Инкапсуляция просто означает скрытие данных. Как правило, в объектно-ориентированном программировании один класс не должен иметь прямого доступа к данным другого класса. Вместо этого, доступ должен контролироваться через методы класса. Чтобы предоставить контролируемый доступ к данным класса в Python, используются модификаторы доступа и свойства. Атрибут может быть объявлен приватным (внутренним) с помощью нижнего подчеркивания перед именем, но настоящего скрытия на самом деле.

Подробную информацию об ООП можно прочитать в источниках, указанных в списке использованной и рекомендуемой литературы.

Основные особенности реализации ООП на Python:

- Классы в Python – это тоже объекты.
- Допустимо динамическое изменение и добавление атрибутов классов.
- Жизненным циклом объекта можно управлять.
- Многие операторы могут быть перезагружены.
- Многие методы встроенных объектов можно эмулировать.
- Для скрытия внутренних данных используются синтаксические соглашения.
- Поддерживается наследование.
- Полиморфизм обеспечивается виртуальностью всех методов.

10.2 Вопросы для текущего контроля успеваемости

- 1) Что такое объектно-ориентированное программирование (ООП)?
- 2) Дайте характеристику классу и экземпляру класса. Приведите примеры.
- 3) Перечислите и охарактеризуйте основные механизмы ООП.
- 4) Что такое атрибуты и методы класса? Приведите примеры.
- 5) Что такое конструктор класса? Зачем он используется?

11 ЛАБОРАТОРНЫЕ РАБОТЫ

Лабораторная работа № 1

Разработка алгоритмов с помощью блок-схем и на словесном языке

- 1 Приведите контрольные примеры для решения задач (по вариантам).
- 2 Составьте алгоритмы для каждой задачи:
 - а) на словесном языке;
 - б) в виде блок-схем.

Вариант 1

1. Даны длины сторон треугольника (a, b, c). Найти площадь треугольника (S), радиусы его вписанной (r) и описанной окружностей (R).
2. Дана функция $f(x) = 2x\sqrt{x} - (1 - 5x)^3$. Найти $f'(a)$ при заданном значении a .
3. Вычислить значение кусочно-заданной функции при заданном значении аргумента (x):

$$y = \begin{cases} (1 + cx^2)^4, & x \leq 1, \\ \sin \frac{2\pi k}{x}, & 1 < x \leq 3, \\ \sqrt[3]{2x - 1}, & x > 3, \end{cases}$$

где $c = -2$; $k = 1.5$.

Вариант 2

1. Даны длины двух сторон треугольника (a, b) и угол между ними (A). Найти площадь треугольника (S) и его третью сторону (c).
2. Дана функция $f(x) = (x + 5)^3(8 - x) + \lg x$. Найти $f'(a)$ при заданном значении a .
3. Вычислить значение кусочно-заданной функции при заданном значении аргумента (x):

$$y = \begin{cases} 2 \cdot \sqrt[3]{x^4 - 1}, & x \leq -1, \\ \sin \sqrt{|ax|} + 2x, & -1 < x \leq 1, \\ e^{\frac{a}{x}} + \lg(x + \sin x), & x > 1, \end{cases}$$

где $a = 1.45$

Вариант 3

1. Даны длина стороны треугольника (a) и два угла (A, B). Найти радиус его описанной окружности (R) и две другие стороны (b, c).
2. Дана функция $f(x) = \frac{18}{x} - (7x - 5)^4 + 4\sqrt{3x + 1}$. Найти $f'(a)$ при заданном значении a .

3. Вычислить значение кусочно-заданной функции при заданном значении аргумента (x):

$$y = \begin{cases} \frac{2 \sin(3x - 1)}{x}, & x < -\pi/2, \\ e^{-x^2} + \cos 2kx, & -\pi/2 \leq x \leq 1, \\ \log_2(3x + k), & x > 1, \end{cases}$$

где $k=1.5$.

Вариант 4

1. Даны длины сторон треугольника (a, b, c). Найти его медианы (ma, mb, mc).
2. Дана функция $f(x) = 11 - (2x - 9)^2 e^{5-x}$. Найти $f'(a)$ при заданном значении a .
3. Вычислить значение кусочно-заданной функции при заданном значении аргумента (x):

$$y = \begin{cases} 2x^3 + |x - 1|, & x < 1, \\ 2a \cos(\pi x - 2), & 1 \leq x \leq 4, \\ a \lg x + \sqrt[3]{x^2}, & x > 4, \end{cases}$$

где $a = -4.2$.

Вариант 5

1. Даны длины двух смежных сторон параллелограмма (a, b) и одной из его диагоналей ($d1$). Найти другую диагональ ($d2$) и высоты (ha, hb).
2. Дана функция $f(x) = (4x^3 + 6x - 7)e^{5-3x} - \frac{4}{x}$. Найти $f'(a)$ при заданном значении a .
3. Вычислить значение кусочно-заданной функции при заданном значении аргумента (x):

$$y = \begin{cases} \frac{4x - 8}{x^4 + k}, & x \leq -2, \\ \cos^2 x + kx^4, & -2 < x \leq k, \\ \sin nx \cdot \ln(2x - k^2), & x > k, \end{cases}$$

где $n=2.8; k=1.25$.

Вариант 6

1. Дана длина стороны правильного n -угольника (a). Найти радиусы его вписанной (r) и описанной окружностей (R) и площадь n -угольника (S).
2. Дана функция $f(x) = (2x - 7)^3(4 - 3x) + 5\sqrt{x}$. Найти $f'(a)$ при заданном значении a .

3. Вычислить значение кусочно-заданной функции при заданном значении аргумента (x):

$$y = \begin{cases} \sin^3(5x - a), & x \leq 0, \\ \frac{e^{-ax}}{ax + 1}, & 0 < x \leq 1, \\ 2\lg(3x - 1), & x > 1, \end{cases}$$

где $a=4.6$.

Вариант 7

1. Дан радиус окружности (R). Найти периметр (p) и площадь (S) правильного n -угольника, вписанного в окружность заданного радиуса.
2. Дана функция $f(x) = 5x\sqrt{x} - \lg(2x - 7)^2$. Найти $f'(a)$ при заданном значении a .
3. Вычислить значение кусочно-заданной функции при заданном значении аргумента (x):

$$y = \begin{cases} \sin^2(2x + a), & x < 1, \\ ax^2 - |3 - x^a|, & 1 \leq x \leq 2, \\ \frac{1}{e^{ax}} \cos \pi x, & x > 2, \end{cases}$$

где $a=1.02$.

Вариант 8

1. Дан радиус окружности (r). Найти периметр (p) и площадь (S) правильного n -угольника, описанного около окружности заданного радиуса, а также радиус окружности, описанной около этого n -угольника.
2. Дана функция $f(x) = x \cdot 2^{5x^2 - 2x - 4} - 6x^2$. Найти $f'(a)$ при заданном значении a .
3. Вычислить значение кусочно-заданной функции при заданном значении аргумента (x):

$$y = \begin{cases} \cos^2 x - ax^3, & x \leq -2, \\ \sqrt{ax^2 + b \sin x + 5}, & -2 < x < 4, \\ \frac{ax + b}{2x - 7}, & x \geq 4, \end{cases}$$

где $a=4.7$; $b=1.5$.

Вариант 9

1. Даны длины оснований трапеции (a , b) и радиус описанной окружностей (R), центр которой находится внутри трапеции. Найти ее высоту (h) и боковые стороны (c , d).

2. Дана функция $f(x) = 5\cos(3 - 2x) - 9\sqrt{x+1} + 12$. Найти $f'(a)$ при заданном значении a .
3. Вычислить значение кусочно-заданной функции при заданном значении аргумента (x):

$$y = \begin{cases} 2ax + |a - x^4|, & x < 0, \\ \frac{e^x}{\sqrt{10 + ax^2}} - 1, & 0 \leq x < 3, \\ 4x^3 - 2x^2 + 5, & x \geq 3, \end{cases}$$

где $a=2.3$.

Вариант 10

1. Даны длины сторон треугольника (a, b, c). Найти площадь треугольника и отрезки (a_1, a_2, b_1, b_2), на которые биссектрисы углов треугольника разбивают его стороны a и b .
2. Дана функция $f(x) = \frac{15}{\sqrt{x}} - 7\sin(14 - 5x) + 3$. Найти $f'(a)$ при заданном значении a .
3. Вычислить значение кусочно-заданной функции при заданном значении аргумента (x):

$$y = \begin{cases} \pi x^3 - \frac{a}{x^4}, & x < 0, \\ \log_3(x + a\sqrt{2x+1} + c), & 0 \leq x \leq 10, \\ \frac{ax - c}{x + 2}, & x > 10, \end{cases}$$

где $a=7.2; c=1$.

Вариант 11

1. Дана длина диагоналей ромба (d_1, d_2). Найти его сторону (a), высоту (h) и синус угла ($\sin A$).
2. Дана функция $f(x) = (3x^4 + 5x^3 - 12x - 8)(3 - 2x) - \sqrt{x^4 - 1}$. Найти $f'(a)$ при заданном значении a .
3. Вычислить значение кусочно-заданной функции при заданном значении аргумента (x):

$$y = \begin{cases} x + \sqrt[3]{x - a}, & x < -1, \\ e^{-ax} \cos(1 - ax^2), & -1 \leq x \leq a - 1, \\ \sqrt{x + 2\sin x}, & x > a - 1, \end{cases}$$

где $a=3.4$.

Вариант 12

1. Даны длины оснований трапеции (a, b) и радиус описанной окружностей (R), центр которой находится вне трапеции. Найти ее высоту (h) и боковые стороны (c, d).
2. Дана функция $f(x) = \left(\frac{3}{2} - 5x\right)\cos(2x) + \sin^2 x$. Найти $f'(a)$ при заданном значении a .
3. Вычислить значение кусочно-заданной функции при заданном значении аргумента (x):

$$z = \begin{cases} \sin \lg ax - |1 - 2^x|, & x < 0, \\ x^2 + \sqrt{x - a}, & 0 \leq x \leq 3, \\ \frac{e^{ax-1}}{1 + x^4}, & x > 3, \end{cases}$$

где $a = -3$.

Лабораторная работа № 2

Знакомство с языком программирования Питон

- 1 Запустите программную среду языка Питон.
- 2 При помощи командной строки решите задачи.
 - a) Вычислить значения факториала: $15! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot 15$.
 - b) $0.3333333333333333 * 3$
 - c) Среди нескольких чисел найти и вывести наименьшее (\min) и наибольшее (\max) значение.
 - d) Определить, является ли натуральное число n нечетным трехзначным числом (вывести True или False).
- 3 Запустите текстовый редактор среды Питон.
- 4 Составьте контрольные примеры и программы для решения задач.
 - a) К финалу конкурса были допущены трое претендентов. Соревнования проходили в три тура. Первый конкурсант набрал в первом туре $m1$ баллов, во втором – $n1$ баллов, в третьем – $p1$ баллов. Второй конкурсант – соответственно $m2, n2, p2$. Третий – $m3, n3, p3$. Определить, сколько баллов набрал победитель.
 - b) Вычислить и вывести сумму цифр данного четырехзначного числа.
- 5 Сохраните в файлы под именами Lab_2_1.py и Lab_2_2.py в своей папке.
- 6 Добавьте к программному коду комментарии. Сохраните изменения.
- 7 Запустите на выполнение и отладьте программы.
- 8 Объясните каждую строчку программы.

Линейные алгоритмы

- 1 Запустите программную среду языка Питон.
- 2 Подключите модуль math (`import math`) и выполните вычисления:
 - a) $258 \cdot \sin 15^\circ$;
 - b) $\cos x + \frac{a-1}{\sqrt{1-x^2}}$, где $a=3 - const$;
 - c) $\sin^2(xy) + \sqrt{x+y}$.
- 3 Введите программный (исходный) код, реализующий алгоритмы решения задач 1 и 2 из лабораторной работы №1.
- 4 Сохраните в файлы под именами Lab_3_1.py и Lab_3_2.py в своей папке.
- 5 Добавьте к программному коду комментарии. Сохраните изменения.
- 6 Запустите на выполнение и отладьте программы.
- 7 Протестируйте работу программ:
 - a) приведите наборы исходных данных и результаты, при которых программа работает правильно;
 - b) а также наборы данных, при которых выводится сообщение об ошибке. Объясните причину.
- 8 Объясните каждую строчку программы.

Функции пользователя

- 1 Опишите функцию для решения задачи и протестируйте ее работу.
 - a) Дано двузначное число. Выведите его цифры через запятую.
 - b) Найти сумму цифр трехзначного числа.
- 2 Составьте алгоритм решения задачи и программу, согласно варианту. Сохраните в файл под именем Lab_4_1.py.
- 3 Используя данные из задачи 2 лабораторной работы №1 и программный код из файла Lab_3_2.py, составьте программу для реализации следующих действий (сохраните под именем Lab_4_2.py):
 - a) Описать функции для вычисления $y = f(x)$ и $y' = f'(x)$.
 - b) Составить уравнение касательной в точке с абсциссой a .
 - c) Вывести координаты точки касания.
- 4 Отладьте и протестируйте программы.

Вариант 1

Три точки в пространстве заданы своими координатами: $A(x_1, y_1, z_1)$, $B(x_2, y_2, z_2)$, $C(x_3, y_3, z_3)$. Описать функцию для вычисления и вывода координат вектора по двум заданным точкам (начало и конец вектора). Вычислить координаты векторов \overrightarrow{AB} , \overrightarrow{BC} и \overrightarrow{CA} .

Вариант 2

Три точки в пространстве заданы своими координатами: $A(x_1, y_1, z_1)$, $B(x_2, y_2, z_2)$, $C(x_3, y_3, z_3)$. Описать функцию для вычисления и вывода длины отрезка по двум заданным точкам. Вычислить длины отрезков AB , BC и AC .

Вариант 3

Три вектора в пространстве заданы своими координатами: $\vec{a}(x_1, y_1, z_1)$, $\vec{b}(x_2, y_2, z_2)$, $\vec{c}(x_3, y_3, z_3)$. Описать функцию для вычисления и вывода скалярного произведения двух векторов. Вычислить скалярные произведения векторов $\vec{a} \cdot \vec{b}$, $\vec{b} \cdot \vec{c}$ и $\vec{a} \cdot \vec{c}$.

Вариант 4

Четыре точки на плоскости заданы своими координатами: $A(x_1, y_1)$, $B(x_2, y_2)$, $C(x_3, y_3)$ и $D(x_4, y_4)$. Описать функцию для вычисления и вывода координат вектора по двум заданным точкам (начало и конец вектора). Вычислить координаты векторов \vec{AB} , \vec{BC} , \vec{CD} и \vec{DA} .

Вариант 5

Четыре точки на плоскости заданы своими координатами: $A(x_1, y_1)$, $B(x_2, y_2)$, $C(x_3, y_3)$ и $D(x_4, y_4)$. Описать функцию для вычисления и вывода длины отрезка по двум заданным точкам. Вычислить длины отрезков AB , BC , CD и AD .

Вариант 6

Четыре вектора на плоскости заданы своими координатами: $\vec{a}(x_1, y_1)$, $\vec{b}(x_2, y_2)$, $\vec{c}(x_3, y_3)$, $\vec{d}(x_4, y_4)$. Описать функцию для вычисления и вывода скалярного произведения двух векторов. Вычислить скалярные произведения векторов $\vec{a} \cdot \vec{b}$, $\vec{b} \cdot \vec{c}$, $\vec{c} \cdot \vec{d}$ и $\vec{a} \cdot \vec{d}$.

Вариант 7

Три точки в пространстве заданы своими координатами: $A(x_1, y_1, z_1)$, $B(x_2, y_2, z_2)$, $C(x_3, y_3, z_3)$. Описать функцию для вычисления и вывода координат вектора по двум заданным точкам (начало и конец вектора). Вычислить координаты векторов \vec{AB} , \vec{BC} и \vec{CA} .

Вариант 8

Три точки в пространстве заданы своими координатами: $A(x_1, y_1, z_1)$, $B(x_2, y_2, z_2)$, $C(x_3, y_3, z_3)$. Описать функцию для вычисления и вывода длины отрезка по двум заданным точкам. Вычислить длины отрезков AB , BC и AC .

Вариант 9

Три вектора в пространстве заданы своими координатами: $\vec{a}(x_1, y_1, z_1)$, $\vec{b}(x_2, y_2, z_2)$, $\vec{c}(x_3, y_3, z_3)$. Описать функцию для вычисления и вывода скалярного произведения двух векторов. Вычислить скалярные произведения векторов $\vec{a} \cdot \vec{b}$, $\vec{b} \cdot \vec{c}$ и $\vec{a} \cdot \vec{c}$.

Вариант 10

Четыре точки на плоскости заданы своими координатами: $A(x_1, y_1)$, $B(x_2, y_2)$, $C(x_3, y_3)$ и $D(x_4, y_4)$. Описать функцию для вычисления и вывода

координат вектора по двум заданным точкам (начало и конец вектора).
Вычислить координаты векторов \overrightarrow{AB} , \overrightarrow{BC} , \overrightarrow{CD} и \overrightarrow{DA} .

Вариант 11

Четыре точки на плоскости заданы своими координатами: $A(x_1, y_1)$, $B(x_2, y_2)$, $C(x_3, y_3)$ и $D(x_4, y_4)$. Описать функцию для вычисления и вывода длины отрезка по двум заданным точкам. Вычислить длины отрезков AB , BC , CD и AD .

Вариант 12

Четыре вектора на плоскости заданы своими координатами: $\vec{a}(x_1, y_1)$, $\vec{b}(x_2, y_2)$, $\vec{c}(x_3, y_3)$, $\vec{d}(x_4, y_4)$. Описать функцию для вычисления и вывода скалярного произведения двух векторов. Вычислить скалярные произведения векторов $\vec{a} \cdot \vec{b}$, $\vec{b} \cdot \vec{c}$, $\vec{c} \cdot \vec{d}$ и $\vec{a} \cdot \vec{d}$.

Лабораторная работа №5

Разветвляющиеся алгоритмы

- 1 Составьте алгоритм решения задачи и опишите его на языке Python:
 - а) Используя признак делимости, проверить, делится ли натуральное число k на 5 (последняя цифра 0 или 5).
 - б) Даны длины трех отрезков (a , b , c – положительные числа). Определить, могут ли они являться сторонами равностороннего или равнобедренного треугольника.
- 2 Составьте алгоритм решения задачи и программу, согласно варианту (проверку условия и вывод результатов оформите в виде функции). Сохраните в файл под именем Lab_5_1.py.
- 3 Используя программный код из файла Lab_3_1.py, составьте решение задачи таким образом, чтобы оно было более универсальным (сохранить под именем Lab_5_2.py):
 - а) добавьте проверку корректности вводимых данных;
 - б) при невыполнении условия добавьте вывод текста о причине, по которой невозможно произвести вычисления.
- 4 Введите программный (исходный) код, реализующий алгоритм решения задачи 3 из лабораторной работы №1.
- 5 Сохраните в файл под именем Lab_5_3.py и запустите на выполнение.
- 6 Добавьте к программному коду комментарии. Сохраните изменения.
- 7 Отладьте и протестируйте программы.

Вариант 1

Используя признаки делимости, для натурального числа k определить, делится ли оно на 5 или делится на 2.

Вариант 2

По введенному названию месяца определить и вывести количество дней в нем.

Вариант 3

По введенному номеру месяца определить и вывести название времени года.

Вариант 4

По введенному названию времени года определить и вывести названия месяцев, относящихся к нему.

Вариант 5

Дана градусная мера угла на плоскости. Определить к какому типу относится данный угол (острый, тупой, прямой или развернутый).

Вариант 6

По введенному номеру недели и названию дня недели вывести количество пар в этот день или сообщение "выходной день".

Вариант 7

Дано натуральное число. Определить, является ли введенное число однозначным, двузначным, трехзначным, четырехзначным или содержит большее количество цифр.

Вариант 8

По введенному с клавиатуры символу определить, к какой категории он относится (буква русского алфавита, буква английского алфавита, цифра или специальный символ).

Вариант 9

Дано целое число k (0-99), определяющее возраст. Вывести сообщение, добавляющее к значению k слово "год", "года" или "лет". Например, при $k=23$ вывести "23 года".

Вариант 10

По введенному номеру курса вывести (в столбик) список групп на этом курсе в нашем институте.

Вариант 11

По введенной температуре воды в градусах Цельсия определить ее агрегатное состояние (твердое, жидкое, газообразное) или вывести сообщение, что такое значение недопустимо.

Вариант 12

Даны два целых числа и символ из множества (+, -, *, /, %). Вывести результат выполнения соответствующей операции над данными числами.

Лабораторная работа №6

Циклические алгоритмы (цикл for)

- 1 Используя оператор цикла *for*, составьте алгоритмы и программы для решения задач:
 - a) Вычислить $S = 2 + 5 + 8 + 11 + \dots + 35$ и сравнить вычислениями, полученными по формуле для суммы элементов арифметической прогрессии.
 - b) Вычислить $a + 2a^2 + 3a^3 + \dots + na^n$, где n , a – целые числа, введенные с клавиатуры.

- 2 Составьте алгоритм решения задач и программы, согласно варианту (вычисление суммы описать в виде функции). Сохраните в файлы под именами Lab_6_1.py и Lab_6_2.py соответственно.
- 3 Отладьте и протестируйте программы.

Вариант 1

1. Вычислить сумму ряда: $\sum_{k=1}^n \frac{3}{(2k-1)(2k+4)}$.
2. Дана последовательность целых чисел. Найти сумму нечетных элементов последовательности, больших -10 , и их количество.

Вариант 2

1. Вычислить сумму ряда: $\sum_{k=1}^n \frac{k}{(k+1)^2 + 3}$.
2. Дана последовательность целых чисел. Найти среднее арифметическое четных отрицательных элементов последовательности.

Вариант 3

1. Вычислить сумму ряда: $\sum_{k=1}^n \frac{2k}{(k+1)(k+2)}$.
2. Дана последовательность целых чисел. Найти произведение нечетных элементов последовательности, меньших 5 , и их количество.

Вариант 4

1. Вычислить сумму ряда: $\sum_{k=1}^n \frac{k+1}{k\sqrt{k^2+2}}$.
2. Дана последовательность целых чисел. Найти среднее арифметическое отрицательных элементов последовательности, кратных 3 .

Вариант 5

1. Вычислить сумму ряда: $\sum_{k=1}^n \frac{k^2}{2^k + 3}$.
2. Дана последовательность целых чисел. Найти сумму положительных элементов последовательности, кратных 3 , и их количество.

Вариант 6

1. Вычислить сумму ряда: $\sum_{k=1}^n \frac{k-2}{k^2 + 3k + 4}$.
2. Дана последовательность целых чисел. Найти сумму элементов последовательности, кратных 3 , меньших 20 , и их количество.

Вариант 7

1. Вычислить сумму ряда: $\sum_{k=1}^n \frac{9k}{2k^2 - 3k + 4}$.
2. Дана последовательность целых чисел. Найти среднее арифметическое положительных элементов последовательности, кратных 5 .

Вариант 8

1. Вычислить сумму ряда: $\sum_{k=1}^n \frac{k+4}{(3k+2)(k+8)}$.

2. Дана последовательность целых чисел. Найти сумму элементов последовательности, кратных 5, больших -10 , и их количество.

Вариант 9

1. Вычислить сумму ряда: $\sum_{k=1}^n \frac{1}{\sqrt{2k^2+7}}$.

2. Дана последовательность целых чисел. Найти сумму нечетных отрицательных и кратных 3 положительных элементов последовательности.

Вариант 10

1. Вычислить сумму ряда: $\sum_{k=1}^n \frac{5k^2}{k^4+4k-1}$.

2. Дана последовательность целых чисел. Найти среднее арифметическое нечетных положительных элементов и среднее арифметическое отрицательных элементов последовательности.

Вариант 11

1. Вычислить сумму ряда: $\sum_{k=1}^n \frac{\sqrt{k}}{(k^2+1)(3k-2)}$.

2. Дана последовательность целых чисел. Найти сумму нечетных положительных и произведение четных отрицательных элементов последовательности.

Вариант 12

1. Вычислить сумму ряда: $\sum_{k=1}^n \frac{k+1}{\sqrt{k^2-4k+5}}$.

2. Дана последовательность целых чисел. Найти произведение четных положительных и сумму нечетных отрицательных элементов последовательности.

Лабораторная работа №7

Циклические алгоритмы (цикл while)

1. Используя оператор цикла *while*, составьте алгоритмы и программы для решения задач:

а) Вычислить сумму ряда $\sum_{k=1}^{\infty} \frac{k}{2^{k+1}+1}$ с заданной точностью *eps*.

б) Вычислить $(n)!!$, где при $n=2k$: $n!!=(2k)!!=2 \cdot 4 \cdot \dots \cdot (2k)$,
при $n=2k-1$: $n!!=(2k-1)!!=1 \cdot 3 \cdot \dots \cdot (2k-1)$.

2. Составьте алгоритм решения задач и программы, согласно варианту. Сохраните в файлы под именами Lab_7_1.py, Lab_7_2.py и Lab_7_3.py соответственно.

3 Отладьте и протестируйте программы.

Вариант 1

1. Дано натуральное число. Определить количество значащих нулей в представлении числа в двоичной системе счисления.
2. Составить таблицу значений функции $y = \ln(x^2 + 1) - (x - 3)^4$ на отрезке $[-2; 4]$ с шагом изменения аргумента $dx=0.5$.
3. Вычислить с заданной точностью *eps* значение функции, разложенной в степенной ряд. Сравнить со значением встроеной функции.

$$y = \sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!},$$

где $x \in (-\infty; \infty)$.

Вариант 2

1. Дано натуральное число. Определить количество единиц в представлении числа в двоичной системе счисления.
2. Составить таблицу значений функции $y = e^{2x} - 4x$ на отрезке $[1; 5]$ с шагом изменения аргумента $dx=0.5$.
3. Вычислить с заданной точностью *eps* значение функции, разложенной в степенной ряд. Сравнить со значением встроеной функции.

$$y = \cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{(2n)!},$$

где $x \in (-\infty; \infty)$.

Вариант 3

1. Дано натуральное число. Определить количество значащих нулей в представлении числа в троичной системе счисления.
2. Составить таблицу значений функции $y = \frac{\sqrt{x-3}}{x^2+2}$ на отрезке $[3; 6]$ с шагом изменения аргумента $dx=0.25$.
3. Вычислить с заданной точностью *eps* значение функции, разложенной в степенной ряд. Сравнить со значением встроеной функции.

$$y = e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots = \sum_{n=0}^{\infty} \frac{x^n}{n!},$$

где $x \in (-\infty; \infty)$.

Вариант 4

1. Дано натуральное число. Определить количество единиц в представлении числа в троичной системе счисления.

2. Составить таблицу значений функции $y = \frac{3x^2}{\sqrt{x^4 + 5}}$ на отрезке $[-2; 2]$ с шагом изменения аргумента $dx=0.25$.

3. Вычислить с заданной точностью *eps* значение функции, разложенной в степенной ряд. Сравнить со значением встроеной функции.

$$y = \sinh x = x + \frac{x^3}{3!} + \frac{x^5}{5!} + \frac{x^7}{7!} + \dots = \sum_{n=0}^{\infty} \frac{x^{2n+1}}{(2n+1)!},$$

где $x \in (-\infty; \infty)$.

Вариант 5

1. Дано натуральное число. Определить сумму цифр в представлении числа в троичной системе счисления.

2. Составить таблицу значений функции $y = \lg(2x - 3) + \frac{1}{x^3}$ на отрезке $[2; 6]$ с шагом изменения аргумента $dx=0.4$.

3. Вычислить с заданной точностью *eps* значение функции, разложенной в степенной ряд. Сравнить со значением встроеной функции.

$$y = \cosh x = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \frac{x^6}{6!} + \dots = \sum_{n=0}^{\infty} \frac{x^{2n}}{(2n)!},$$

где $x \in (-\infty; \infty)$.

Вариант 6

1. Дано натуральное число. Определить количество единиц в представлении числа в восьмеричной системе счисления.

2. Составить таблицу значений функции $y = \log_2(2 - 4x) - \sin x$ на отрезке $[-3; -1]$ с шагом изменения аргумента $dx=0.2$.

3. Вычислить с заданной точностью *eps* значение функции, разложенной в степенной ряд. Сравнить со значением встроеной функции.

$$y = e^{-x} = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \dots = \sum_{n=0}^{\infty} \frac{(-1)^n x^n}{n!},$$

где $x \in (-\infty; \infty)$.

Вариант 7

1. Дано натуральное число. Определить количество значащих нулей в представлении числа в восьмеричной системе счисления.

2. Составить таблицу значений функции $y = \sin^2 x - (x + 1)^3$ на отрезке $[-2; 2]$ с шагом изменения аргумента $dx=0.5$.

3. Вычислить с заданной точностью *eps* значение функции, разложенной в степенной ряд. Сравнить со значением встроеной функции.

$$y = \frac{1}{\sqrt{1+x}} = 1 - \frac{1}{2}x + \frac{1 \cdot 3}{2 \cdot 4}x^2 - \frac{1 \cdot 3 \cdot 5}{2 \cdot 4 \cdot 6}x^3 + \dots =$$

$$= 1 + \sum_{n=1}^{\infty} \frac{(-1)^n (2n-1)!!}{(2n)!!} x^n,$$

где $x \in (-1; 1]$.

Вариант 8

1. Дано натуральное число. Определить сумму цифр, меньших 3, в представлении числа в восьмеричной системе счисления.
2. Составить таблицу значений функции $y = \operatorname{tg} x + 2x^2 - 3x$ на отрезке $[-1.5; 1.5]$ с шагом изменения аргумента $dx=0.3$.
3. Вычислить с заданной точностью *eps* значение функции, разложенной в степенной ряд. Сравнить со значением встроеной функции.

$$y = (1+x)^m = 1 + mx + \frac{m(m-1)}{2!}x^2 + \frac{m(m-1)(m-2)}{3!}x^3 + \dots =$$

$$= 1 + \sum_{n=1}^{\infty} \frac{m(m-1)\dots(m-n+1)}{n!}x^n,$$

где $x \in (-1; 1)$.

Вариант 9

1. Дано натуральное число. Определить количество цифр, меньших 3, в представлении числа в восьмеричной системе счисления.
2. Составить таблицу значений функции $y = 3x\sqrt{x^2+1} - 2x^3 + 1$ на отрезке $[-1; 2]$ с шагом изменения аргумента $dx=0.25$.
3. Вычислить с заданной точностью *eps* значение функции, разложенной в степенной ряд. Сравнить со значением встроеной функции.

$$y = a^x = 1 + x \ln a + \frac{(x \ln a)^2}{2!} + \frac{(x \ln a)^3}{3!} + \dots = \sum_{n=0}^{\infty} \frac{(x \ln a)^n}{n!},$$

где $x \in (-\infty; \infty)$.

Вариант 10

1. Дано натуральное число. Определить количество цифр, меньших 2, в представлении числа в троичной системе счисления.
2. Составить таблицу значений функции $y = 3\ln(x^2-1) - 2x$ на отрезке $[2; 4]$ с шагом изменения аргумента $dx=0.25$.
3. Вычислить с заданной точностью *eps* значение функции, разложенной в степенной ряд. Сравнить со значением встроеной функции.

$$y = \arcsin x = x + \frac{1}{2 \cdot 3} x^3 + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5} x^5 + \frac{1 \cdot 3 \cdot 5}{2 \cdot 4 \cdot 6 \cdot 7} x^7 + \dots =$$

$$= x + \sum_{n=1}^{\infty} \frac{(2n-1)!!}{(2n)!! \cdot (2n+1)} x^{2n+1},$$

где $x \in [-1; 1]$.

Вариант 11

1. Дано натуральное число. Определить количество цифр, меньших 2, в представлении числа в пятеричной системе счисления.
2. Составить таблицу значений функции $y = 4\sqrt[3]{x^2} - 2\cos(x-1)$ на отрезке $[-1; 4]$ с шагом изменения аргумента $dx=0.5$.
3. Вычислить с заданной точностью *eps* значение функции, разложенной в степенной ряд. Сравнить со значением встроеной функции.

$$y = \operatorname{arctg} x = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \dots = \sum_{n=1}^{\infty} \frac{(-1)^{n-1} x^{2n-1}}{2n-1},$$

где $x \in [-1; 1]$.

Вариант 12

1. Дано натуральное число. Определить количество значащих нулей в представлении числа в пятеричной системе счисления.
2. Составить таблицу значений функции $y = 2\sin^2(3-x^2) + 2x - 5$ на отрезке $[0; 4]$ с шагом изменения аргумента $dx=0.4$.
3. Вычислить с заданной точностью *eps* значение функции, разложенной в степенной ряд. Сравнить со значением встроеной функции.

$$y = \ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \dots = \sum_{n=0}^{\infty} \frac{(-1)^n x^{n+1}}{n+1},$$

где $x \in (-1; 1]$.

Лабораторная работа №8

Векторы (одномерные массивы)

1. Составьте программы, выполняющие ввод данных пользователя в массив и выполнения следующих действий:
 - a) В одномерном массиве целых чисел найти сумму элементов, больших 5.
 - b) В одномерном массиве вещественных чисел все элементы уменьшить на 1.5.

- 2 Составьте алгоритм решения задач и программы, согласно варианту. Сохраните в файлы под именами Lab_8_1.py, Lab_8_2.py и Lab_8_3.py соответственно.
- 3 Отладьте и протестируйте программы.

Вариант 1

1. Даны два вектора a и b линейного пространства \mathbb{R}^n . Найти $c=1.2a+3.5b$.
2. Ввести произвольные целочисленные данные в первый массив. Создать второй массив по следующему правилу: сначала записать нечетные числа, затем четные числа из первого массива. Порядок их следования не изменять.
3. В одномерном массиве вещественных чисел все элементы, большие нулевого элемента, уменьшить в два раза.

Вариант 2

1. Даны два вектора a и b линейного пространства \mathbb{R}^n . Найти $c=2b-3.7a$.
2. Создать массив целых чисел, в котором первые два элемента вводятся с клавиатуры, а каждый следующий элемент равен сумме всех ему предшествующих.
3. В одномерном массиве вещественных чисел все положительные элементы уменьшить на 10.

Вариант 3

1. Даны два вектора a и b линейного пространства \mathbb{R}^n . Найти $c=7.3a-3.4b$.
2. Ввести произвольные целочисленные данные в первый массив. Создать второй массив по следующему правилу: сначала записать положительные числа и нули, затем отрицательные числа из первого массива. Порядок их следования не изменять.
3. В одномерном массиве вещественных чисел все элементы, меньшие последнего элемента, увеличить в три раза.

Вариант 4

1. Даны два вектора a и b линейного пространства \mathbb{R}^n . Найти $c=3.1a-5.3b$.
2. Создать массив целых чисел, являющихся элементами арифметической прогрессии (начальный элемент и разность прогрессии запросить у пользователя).
3. В одномерном массиве вещественных чисел все отрицательные элементы увеличить на k .

Вариант 5

1. Даны два вектора a и b линейного пространства \mathbb{R}^n . Найти $c=7.8a-0.3b$.
2. Ввести произвольные целочисленные данные в первый массив. Создать второй массив по следующему правилу: сначала записать однозначные

числа (положительные и отрицательные), затем все остальные числа из первого массива. Порядок их следования не изменять.

3. В одномерном массиве вещественных чисел все элементы, кратные 3, умножить на $\pi/2$.

Вариант 6

1. Даны два вектора a и b линейного пространства R^n . Найти $c=2.4b-3.7a$.
2. Создать массив вещественных чисел, в котором два последних элемента вводятся с клавиатуры, а каждый предыдущий элемент равен сумме всех последующих.
3. В одномерном массиве целых чисел все однозначные элементы возвести в 3 степень.

Вариант 7

1. Даны два вектора a и b линейного пространства R^n . Найти $c=2.3a+5.1b$.
2. Ввести произвольные целочисленные данные в первый массив. Создать второй массив по следующему правилу: сначала записать все числа, кратные 3, затем остальные числа из первого массива. Порядок их следования не изменять.
3. В одномерном массиве вещественных чисел все отрицательные элементы возвести в квадрат.

Вариант 8

1. Даны два вектора a и b линейного пространства R^n . Найти $c=3.5a-5.3b$.
2. Создать массив вещественных чисел, являющихся элементами геометрической прогрессии (начальный элемент и знаменатель прогрессии запросить у пользователя).
3. В одномерном массиве целых чисел все нечетные элементы уменьшить на k .

Вариант 9

1. Даны два вектора a и b линейного пространства R^n . Найти $c=2.9a+5.2b$.
2. Ввести произвольные целочисленные данные в первый массив. Создать второй массив по следующему правилу: сначала записать все делители числа 60, затем остальные числа из первого массива. Порядок их следования не изменять.
3. В одномерном массиве вещественных чисел из всех элементов, больших или равных 10, извлечь квадратный корень.

Вариант 10

1. Даны два вектора a и b линейного пространства R^n . Найти $c=2.6b-7.7a$.

2. Создать массив вещественных чисел, в котором элемент a_0 ввести с клавиатуры, а каждый следующий элемент вычислить по формуле $a_{i+1} = \sqrt{a_i} + 2$.
3. В одномерном целых массиве чисел все положительные элементы разделить нацело на k .

Вариант 11

1. Даны два вектора a и b линейного пространства R^n . Найти $c=1.7a-6.3b$.
2. Ввести произвольные целочисленные данные в первый массив. Создать второй массив по следующему правилу: сначала записать нечетные числа, затем нули, количество которых равно количеству нечетных чисел из первого массива. Порядок их следования не изменять.
3. В одномерном массиве вещественных чисел все элементы, меньшие или равные 5, увеличить на значение минимального элемента.

Вариант 12

1. Даны два вектора a и b линейного пространства R^n . Найти $c = 3.6a - 6.5b$.
2. Создать массив целых чисел, в котором элемент a_0 ввести с клавиатуры, а каждый следующий элемент вычислить по формуле $a_{i+1} = 2a_i - 3$.
3. В одномерном массиве вещественных чисел все элементы, начиная с первого отрицательного, обнулить.

Лабораторная работа №9

Матрицы (двумерные массивы)

- 1 Составьте функции, выполняющие следующие действия:
 - a) Ввести данные в матрицу заданной размерности.
 - b) Вычислить определитель квадратной матрицы второго порядка.
 - c) Найти сумму двух матриц.
 - d) Найти произведение матрицы на число.
 - e) Вывести матрицу.
- 2 Составьте алгоритм решения задачи и программу, согласно варианту. Сохраните в файл под именем Lab_9_1.py.
- 3 Отладьте и протестируйте программы.

Вариант 1

Дана матрица размерностью $m \times n$. Вычислить среднее арифметическое элементов первой строки матрицы, прибавить его к последним элементам каждой строки. Вывести измененную матрицу на экран.

Вариант 2

Дана матрица размерностью $n \times n$. Найти первый отрицательный элемент во второй строке матрицы, если он существует, и прибавить его к элементам

главной диагонали, иначе оставить матрицу без изменения. Вывести матрицу на экран.

Вариант 3

Дана матрица размерностью $m \times n$. Вычислить суммы элементов каждого столбца и записать их в одномерный массив (вектор), а затем добавить к данной матрице. Вывести полученный вектор и измененную матрицу на экран.

Вариант 4

Дана матрица размерностью $m \times n$. Найти ее минимальный элемент, вычесть его значение из всех элементов предпоследней строки матрицы. Вывести измененную матрицу на экран.

Вариант 5

Дана матрица размерностью $n \times n$. Найти максимальный элемент главной диагонали, разделить на него (если он положительный) все элементы предпоследнего столбца матрицы, иначе оставить матрицу без изменения. Вывести измененную матрицу на экран.

Вариант 6

Дана матрица размерностью $m \times n$. В каждой строке найти минимальные элементы и записать их в одномерный массив (вектор). Найти сумму этих элементов и прибавить ее ко всем элементам последнего столбца. Вывести полученный вектор и измененную матрицу на экран.

Вариант 7

Дана матрица размерностью $m \times n$. Поменять местами строки матрицы, содержащие минимальный и максимальный элементы. Вывести измененную матрицу на экран.

Вариант 8

Дана матрица размерностью $n \times n$. Найти сумму элементов последнего столбца матрицы и прибавить ее ко всем элементам побочной диагонали. Вывести измененную матрицу на экран.

Вариант 9

Дана матрица размерностью $m \times n$. Вычислить суммы элементов каждой строки и записать их в одномерный массив (вектор). Прибавить элементы этого вектора ко всем элементам первого столбца матрицы соответственно. Вывести полученный вектор и измененную матрицу на экран.

Вариант 10

Дана матрица размерностью $m \times n$. Найти ее максимальный элемент, прибавить его значение ко всем элементам предпоследнего столбца и последней строки матрицы. Вывести измененную матрицу на экран.

Вариант 11

Дана матрица размерностью $n \times n$. Найти максимальный элемент в каждом столбце матрицы и поставить его на место диагонального элемента, соответствующего этому столбцу. Вывести измененную матрицу на экран.

Вариант 12

Дана матрица размерностью $m \times n$. Вычислить среднее геометрическое элементов второго столбца матрицы и умножить на него все элементы последней строки. Вывести измененную матрицу на экран.

Лабораторная работа №10

Словари

- 1 Составьте и заполните таблицу данными в соответствии с вашим вариантом. Задайте два критерия для отбора данных и выполните контрольные примеры.
- 2 Составьте программу (Lab_10.py), в которой должны быть предусмотрены следующие возможности (оформите в виде меню):
 - a) создание массива структур (словарей);
 - b) добавление данных;
 - c) вывод всех данных;
 - d) поиск и вывод данных в зависимости от заданных критериев (не менее двух).
- 3 Отладьте и протестируйте программу.

Вариант 1

Название химического элемента	Символ элемента	Относительная атомная масса
A(11)	A(2)	9(7),9(3)

Вариант 2

ФИО нобелевского лауреата	Страна	Год назначения премии
A(21)	A(14)	9(4)

Вариант 3

Название муз. произведения	Исполнитель	Длительность, мин.
A(20)	A(16)	9(5),9(2)

Вариант 4

ФИО участника соревнований	Учебное заведение	Количество баллов
A(23)	X(10)	9(2)

Вариант 5

Название бытового прибора	Назначение	Цена, р.
A(20)	A(20)	9(8),9(2)

Вариант 6

Инвентаризационный номер	Название объекта	Год поступления
X(10)	A(20)	9(4)

Вариант 7

Название социальной сети	Год запуска	Кол-во аккаунтов (млн. шт)
A(13)	9(4)	9(6),9(1)

Вариант 8

Счетное устройство	Разработчик	Год
X(20)	X(15)	9(4)

Вариант 9

Название блюда	Время приготовления, мин	Сложность
A(15)	9(3)	A(10)

Вариант 10

Медицинский препарат	Дозировка, мг.	Цена, р.
X(15)	9(4)	9(7),9(2)

Вариант 11

Наименование изделия	Дата производства	Количество в партии, шт.
A(20)	X(10)	9(3)

Вариант 12

ФИО студента	Дата рождения	Спортивная секция
A(20)	X(10)	A(15)

Текстовые файлы

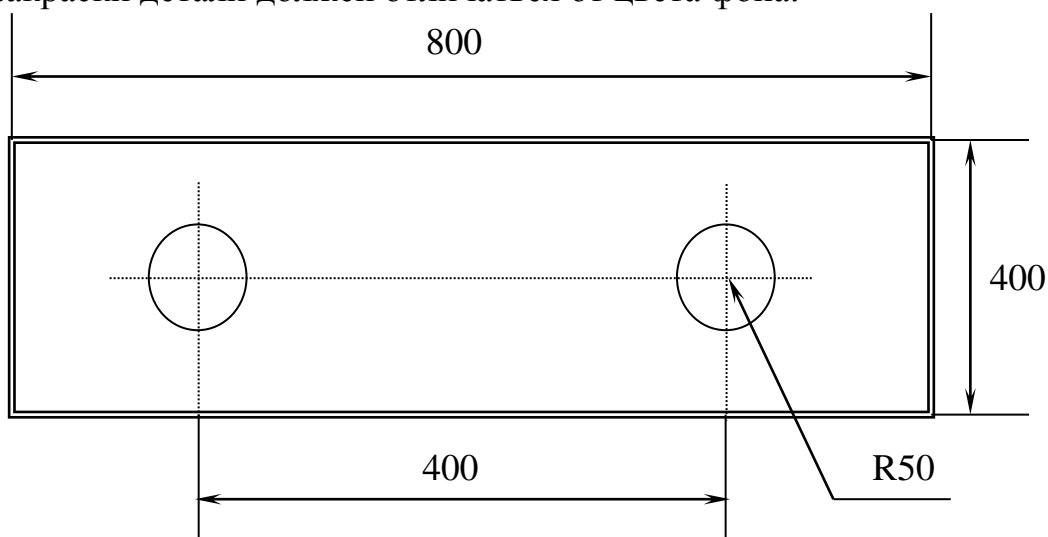
- 1 Заполните текстовый файл данными в соответствии со структурой из лабораторной работы №10.
- 2 Составьте программу (Lab_11.py) для чтения данных из файла, созданного редактором, добавления данных в файл, поиска и вывода данных в соответствии с заданными критериями.
- 3 Вывод данных из файла оформить в виде таблиц с заголовками.
- 4 Отладьте и протестируйте программу.
- 5 Объясните каждую строчку программы.

Файлы прямого доступа

- 1 Составьте программу (Lab_12_1.py) для создания бинарного файла в соответствии со структурой из лабораторной работы №10.
- 2 Составьте программу (Lab_12_2.py) для чтения данных из бинарного файла (созданного предыдущей программой), добавления данных в этот файл, поиска и вывода данных в соответствии с заданными критериями.
- 3 Вывод данных из файла оформить в виде таблиц с заголовками.
- 4 Отладьте и протестируйте программу.
- 5 Объясните каждую строчку программы.

Графика

- 1 Составьте программу построения чертежа детали «ПЛАНКА». Цвет закраски детали должен отличаться от цвета фона.



- 2 Составьте одну из программ:
 - а) Постройте окружность с центром в центре холста. Радиус окружности должен увеличиваться от 10 пикселей до границы холста, затем уменьшаться до 10 пикселей. При этом цвета границы

и внутренней закраски окружности должны периодически изменяться.

- б) Постройте фигуру на холсте и задайте ей движение в нескольких направлениях. При этом фигура должна изменять свои размеры и цвет.

Лабораторная работа №14

Объекты и классы

- 1 Опишите структуру из лабораторной работы №10 в виде класса.
- 2 Создайте оконное приложение с использованием объектов из модуля Tkinter, оформленное в виде пакета.
- 3 Отладьте и протестируйте программу.
- 4 Объясните каждую строчку программы.

12 ЗАДАНИЯ ДЛЯ ПРАКТИЧЕСКИХ ЗАНЯТИЙ И САМОСТОЯТЕЛЬНОЙ РАБОТЫ

12.1 Составление алгоритмов решения задач

- 1 Даны три целых числа (k, m, n) . Найти среднее арифметическое и среднее геометрическое этих чисел.
- 2 Дана длина двух сторон треугольника (a, b) и длина биссектрисы, заключенной между этими сторонами (l) . Найти третью сторону треугольника (c) и площадь (S) .
- 3 Дана функция $f(x) = \frac{1 - \cos 2x}{1 + \cos 2x}$. Найти $f'(a)$ при заданном значении a .
- 4 Вычислить значение кусочно-заданной функции при заданном значении аргумента (x) :

$$y = \begin{cases} 2x + x^3, & x < -3, \\ a - \operatorname{tg} \frac{x}{2}, & -3 \leq x \leq 3, \\ \sqrt{(x+1)^2 - a}, & x > 3, \end{cases}$$

где $a=3.8$.

- 5 Даны три числа. Найти наибольшее из них.

Порядок выполнения каждого задания:

- 1) Привести расчетные формулы.
- 2) Разработать контрольный пример (или примеры).
- 3) Составить алгоритмы:
 - а) на словесном языке;
 - б) в виде блок-схем.

12.2 Выражения

1 Вычислить значения выражений, записанных на языке Python:

- a) $20 / 4$
 $20 // 4$
 $20 \% 4$
- b) $20 / 6$
 $20 // 6$
 $20 \% 6$
- c) $3.0 / 3$
 $3.0 // 3$
 $3.0 \% 3$
- d) $4 / 3.0$
 $4 // 3.0$
 $4 \% 3.0$
- e) $8 // 6 \% 4 * 2$
 $8 // (6 \% 4) * 2$
 $8 \% (6 \% 4 * 2)$
- f) $10 \% 7 + 5 // 4 * 2$
 $10 \% (7 + 5) // 4 * 2$
 $10 \% ((7 + 5) // 4) * 2$
- g) $x = 15 // (8 \% 3)$
 $y = 17 \% x * 5 - 19 \% 5 * 2$
- h) $x = 2 * 5 // 3 \% 2$
 $y = 2 * 5 // (3 \% 2)$
 $x *= y$
 $y = y ** 2$
 $y *= y$
 $y ** = 2$

2 Записать на языке Python следующие выражения:

- a) $x \in (2; 5)$;
- b) $x \notin (2; 5)$;
- c) $x \in [-1; 1]$;
- d) $x \notin [-1; 1]$;
- e) $x \in (-1; 1)$;
- f) хотя бы одно из трех чисел положительно;
- g) ни одно из трех чисел не является положительным;
- h) x^{100} ;
- i) $\sqrt[3]{1+x}$.

3 Определить, равна ли сумма первых двух цифр заданного четырехзначного числа сумме двух его последних цифр.

4 Даны три произвольных числа. Записать выражение, которое позволит определить, можно ли построить треугольник с такими длинами сторон.

- 5 Определить, являются ли три введенных целых числа пифагоровой тройкой, т.е. $a^2+b^2=c^2$.

12.3 Функции и модули

- 1 Записать на языке Python следующие выражения:

a) $\sqrt{1+x^2}$

b) $|a+bx|$

c) $\log_2 x/5$

d) $\cos^2 x^3$

- 2 Составить функцию, определяющую, принадлежит ли точка с координатами (a, b) графику функции $y = 5x^2 - 7x + 2$.

- 3 Даны две тройки целых различных чисел. В каждой из них найти и вывести «среднее» число (не минимальное и не максимальное).

12.4 Разветвляющиеся алгоритмы

- 1 Вычислить значение функции при заданном значении аргумента x .

a) $y = \begin{cases} \sin x, & x > 0, \\ \cos x, & x \leq 0; \end{cases}$

b) $y = \begin{cases} 4x^2, & x < 1, \\ \frac{x^2}{\sqrt[3]{x}}, & 1 \leq x < 5, \\ \ln x, & x \geq 5. \end{cases}$

- 2 Вычислить значение функции, учитывая ее область определения:

$$y = \frac{x+3}{2-x}.$$

- 3 Составить рекурсивную функцию для вычисления:

a) $n!$ – факториала числа;

b) n -го значения в последовательности Фибоначчи ($a_1 = a_2 = 1$, $a_i = a_{i-1} + a_{i-2}$).

- 4 Решить линейное неравенство: $ax < b$.

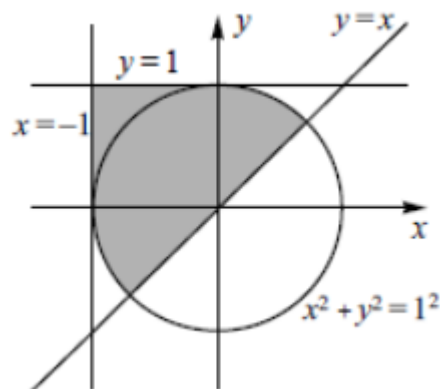
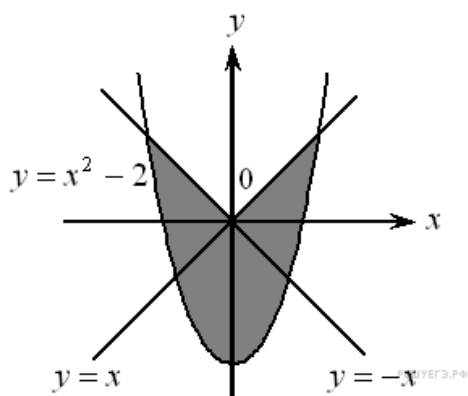
- 5 Решить квадратное уравнение: $ax^2 + bx + c = 0$.

- 6 Каждое из чисел a и b отлично от нуля. Если числа имеют одинаковый знак, то заменить меньшее из них большим. Если числа имеют разные знаки, то присвоить каждому из них знак числа меньшего по абсолютной величине.

- 7 Вычислить стоимость покупки с учетом скидки (описать в виде функции). Если стоимость покупки до 1000 рублей – скидка 5%, до 10000 рублей – 10%, свыше 10000 рублей – 15%.

- 8 Даны координаты точки на плоскости, не лежащей на координатных осях. Определить и вывести номер четверти, которой принадлежит эта точка.

- 9 Определить, принадлежит ли точка с координатами (a, b) закрашенной области:



12.5 Цикл For

- 1 Дана последовательность из n целых чисел. Найти количество элементов последовательности, которые делятся на 5 и не делятся на 7.
- 2 Дана последовательность из n целых чисел. Найти количество пар равных соседних элементов.
- 3 Вычислить значение суммы: $S = \sum_{i=1}^{100} \frac{1}{3i+2}$.
- 4 Вычислить $S = 2^2 + 2^3 + 2^4 + \dots + 2^{10}$. Операцию возведения в степень использовать нельзя.
- 5 Найти все двузначные числа, которые делятся на сумму своих цифр.
- 6 Найти все трехзначные натуральные числа, средняя цифра которых равна сумме первой и третьей цифры.
- 7 Дана последовательность из n целых чисел. Найти два не равных между собой элемента последовательности, которые меньше всех остальных.

12.6 Цикл While

- 1 Составить программу, которая будет вычислять сумму чисел, введенных с клавиатуры, до тех пор, пока сумма не станет больше 30.
- 2 Составить таблицу перевода дюймов в сантиметры, для промежутка от 5 до 10 дюймов с шагом 0.5 дюйма (1 дюйм=2.54 см).
- 3 Составить таблицу значений функции $y = 2 \cdot \sqrt[4]{|x|}$ на отрезке $[-3; 3]$ с шагом изменения аргумента $dx=0.5$.
- 4 Дано натуральное число n . Вычислить сумму и количество цифр, составляющих данное число.
- 5 Дано натуральное число n . Проверить, является ли это число перевертышем (одинаково читается справа налево и слева направо).
- 6 Дано натуральное число n . Разложить его на простые множители.
- 7 Вывести все целые числа, кратные 3, на интервале от k до n , где $k < n$ – целые числа.
- 8 Дано натуральное число n . Вычислить сумму и количество цифр, составляющих данное число.
- 9 Вычислить сумму ряда $\sum_{k=1}^{\infty} \frac{k}{2^{k+1} + 1}$ с заданной точностью eps .

12.7 Векторы (одномерные массивы)

- 1 В одномерном массиве целых чисел найти произведение положительных однозначных элементов массива.
- 2 Даны два массива. Объединить элементы этих массивов в один так, чтобы он остался упорядоченным.
- 3 В одномерном массиве вещественных чисел найти среднее арифметическое отрицательных элементов массива.
- 4 В одномерном массиве, состоящем из n целых чисел, каждый второй элемент обнулить. Измененный массив вывести на экран.
- 5 В одномерном массиве определить наибольшее количество одинаковых соседних элементов.
- 6 В одномерном массиве найти сумму чисел, расположенных между максимальным и минимальным элементами.
- 7 В одномерном массиве поменять местами элементы, так чтобы получить массив в обратном порядке.
- 8 Даны два массива. Найти наименьший элемент первого массива, который не входит во второй массив (считая, что хотя бы один такой элемент есть).

12.8 Двумерные массивы

- 1 Дана матрица размерностью $n \times n$. Вывести на экран элементы побочной диагонали.
- 2 Определить, является ли матрица симметричной относительно главной диагонали.
- 3 Поменять местами максимальный и минимальный элементы матрицы.
- 4 Поэлементно вычесть последний столбец из всех столбцов, кроме последнего.
- 5 Решить систему из трех линейных уравнений методом Крамера.
- 6 Решить совместную определенную систему линейных уравнений методом Гаусса.
- 7 Найти определитель четвертого порядка разложением по строке (или столбцу).

12.9 Файлы

- 8 Дан текстовый файл. Определить длину строки с наибольшим количеством символов. Вывести из файла все строки максимальной длины.
- 9 Записать в текстовый файл таблицу перевода дюймов в сантиметры для промежутка от 1 до 6 см шагом 0.5 см (1 дюйм = 2.54 см).
- 10 В текстовом файле записана матрица. Найти и вывести ее наибольший и наименьший элементы.
- 11 В текстовом файле содержится матрица. Считать данные из файла в двумерный список. Поменять местами 2 и 5 строки, 1 и 3 столбцы. Измененную матрицу записать в другой текстовый файл.
- 12 Ввести с клавиатуры n целых чисел размером 4 байта в знаковом формате. Записать эти числа в двоичный файл.

13 Создать двоичный файл, состоящий из вещественных чисел, вводимых с клавиатуры. Переписать числа в файле в обратном порядке (новый файл не создавать, список не использовать).

13 ВОПРОСЫ ДЛЯ ПРОМЕЖУТОЧНОЙ АТТЕСТАЦИИ

- 1 Алгоритмы: определение, свойства, типы алгоритмических структур.
- 2 Программирование. Уровни и поколения языков программирования. Трансляторы.
- 3 Синтаксис языка Python (физические и логические строки, отступы, комментарии). Интерактивный и файловый режимы обработки команд.
- 4 Операторы и выражения.
- 5 Идентификаторы и объекты (на примере переменных).
- 6 Ввод данных разного типа.
- 7 Форматированный вывод данных.
- 8 Стандартные классы (целый, вещественный, строковый) и константы. Характеристики констант: тип, размер, адрес в ОП.
- 9 Модули и функции. Импорт модулей.
- 10 Стек вызовов. Пространство имен и область видимости.
- 11 Назначение функции. Описаний функции пользователя. Способы указания параметров функции.
- 12 Локальные, нелокальные и глобальные переменные.
- 13 Рекурсивные функции.
- 14 Условный оператор.
- 15 Цикл for.
- 16 Цикл while, управляющие операторы.
- 17 Структуры данных: список и кортеж.
- 18 Структуры данных: словарь, множество.
- 19 Текстовый файл.
- 20 Двоичный файл.
- 21 Работа с графикой. Движение графических объектов
- 22 Объекты и классы.
- 23 Объектно-ориентированное программирование.

СПИСОК ИСПОЛЬЗОВАННОЙ И РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. ГОСТ 19.701-90 (ISO 5807-85) Единая система программной документации. Схемы алгоритмов программ, данных и систем. Условные обозначения и правила выполнения.
2. Официальный сайт Python – Режим доступа: <https://www.python.org/>
3. Балджи, А.С. Математика на Python: учебно-методическое пособие / А.С. Балджи, М.Б. Хрипунова, И.А. Александрова; Финансовый университет при Правительстве Российской Федерации. – Москва: Прометей, 2018. – Ч. 1. Элементы линейной алгебры и аналитической геометрии. – 76 с. : табл. –

Библиогр. в кн. – ISBN 978-5-907003-86-6; То же [Электронный ресурс]. – URL: <http://biblioclub.ru/index.php?page=book&id=494849>

4. Задачи по программированию [Электронный ресурс]: учебное пособие / С.М. Окулов [и др.]. – Электрон. дан. – Москва: Издательство "Лаборатория знаний", 2017. – 826 с. – Режим доступа: <https://e.lanbook.com/book/94162>

5. Златопольский, Д.М. Основы программирования на языке Python [Электронный ресурс]: учебник / Д.М. Златопольский. – Электрон. дан. – Москва: ДМК Пресс, 2017. – 284 с. – Режим доступа: <https://e.lanbook.com/book/97359>

6. Лучано, Р. Python. К вершинам мастерства [Электронный ресурс] / Р. Лучано; пер. с англ. А.А. Слинкин. – Электрон. дан. – Москва: ДМК Пресс, 2016. – 768 с. – Режим доступа: <https://e.lanbook.com/book/93273>

7. Северенс, Ч. Введение в программирование на Python / Ч. Северенс. – 2-е изд., испр. – Москва: Национальный Открытый Университет «ИНТУИТ», 2016. – 231 с.: схем., ил.; То же [Электронный ресурс]. – URL: <http://biblioclub.ru/index.php?page=book&id=429184>

8. Шелудько, В.М. Основы программирования на языке высокого уровня Python: учебное пособие / В.М. Шелудько; Министерство науки и высшего образования РФ, Федеральное государственное автономное образовательное учреждение высшего образования «Южный федеральный университет», Инженерно-технологическая академия. – Ростов-на-Дону; Таганрог: Издательство Южного федерального университета, 2017. – 147 с.: ил. – Библиогр. в кн. – ISBN 978-5-9275-2649-9; То же [Электронный ресурс]. – URL: <http://biblioclub.ru/index.php?page=book&id=500056>

9. Шелудько, В.М. Язык программирования высокого уровня Python: функции, структуры данных, дополнительные модули: учебное пособие / В.М. Шелудько; Министерство науки и высшего образования РФ, Федеральное государственное автономное образовательное учреждение высшего образования «Южный федеральный университет», Институт компьютерных технологий и информационной безопасности. – Ростов-на-Дону; Таганрог: Издательство Южного федерального университета, 2017. – 108 с.: ил. – Библиогр. в кн. – ISBN 978-5-9275-2648-2; То же [Электронный ресурс]. – URL: <http://biblioclub.ru/index.php?page=book&id=500060>