

Глава 1

Язык Python и его особенности

1.1 Общие сведения о языке

Эта глава полезна для общего развития, её можно прочитать, но совершенно необязательно сразу пытаться понять. Лучше вернуться к ней ещё раз после выполнения всех практических заданий.

В основном Python используется как интерпретатор, хотя существуют средства компиляции Python-программ.

Интерпретатор Python позволяет выполнять команды и операции средствами интерактивной оболочки Python (см. далее).

Python — объектно-ориентированный язык. Программные модули и структуры данных могут использоваться как объекты, т.е. имеют свойства и методы.

Подпрограммы в Python оформляются только как функции, но эти функции могут не возвращать значений, а могут возвращать несколько значений в виде структур данных.

Функции, оформленные по определённым правилам, объединяются в модули (библиотеки функций). Модули (или некоторые функции из модулей) по мере необходимости подключаются к пользовательским программам. Такой подход позволяет экономить память вычислительной системы и не занимать её ненужным кодом.

Модули подключаются в начале программы с помощью команды:

```
import имя_модуля
```

А отдельные функции — с помощью команды:

```
from имя_модуля import функция1, ... функцияN
```

Присваивание в Python обозначается знаком «=», а равенство — знаком «==».

В Python разрешены «цепочки» присваиваний и сравнений, однако присваивания и сравнения нельзя смешивать. Выражения $a=b=c=4$ и $a<b<5$ допустимы, а выражение $a<b=4$ — недопустимо. Однако допустимо выражение $a<b==4$.

В Python отсутствуют «операторные скобки» типа `begin ... end` или `DO ... LOOP`. Вместо них в составных операторах (ветвления, циклы, определения функций) используются отступы от начала строки (пробелы).

И наконец, элементы структуры данных могут иметь отрицательные номера.

В следующих главах этой части даётся неполное и полужормальное введение в Python. Этого материала достаточно для выполнения заданий в пределах школьного курса. Для желающих узнать больше имеется список источников информации (раздел «Литература») и справка Python.

1.2 Типы и структуры данных

1.2.1 Типы данных

1.2.1.1 Числа

Числа в Python могут быть обычными целыми (тип `int`), длинными целыми (тип `long`), вещественными (тип `float`) и комплексными (они не будут рассматриваться и использоваться). Для всех задач в пределах школьного курса используются только целые и вещественные числа.

Для преобразования чисел из вещественных в целые и наоборот в Python определены функции `int()` и `float()`. Например, `int(12.6)` даст в результате 12, а `float(12)` даёт в результате 12.0 (десятичный разделитель — точка). Основные операции с числами приведены в таблице 1.1.

Таблица 1.1: Операции с числами

Операция	Описание
$x + y$	Сложение (сумма x и y)
$x - y$	Вычитание (разность x и y)
$x * y$	Умножение (произведение x и y)
x/y	Деление x на y (частное). Внимание! Если x и y целые, то результат всегда будет целым числом! Для получения вещественного результата хотя бы одно из чисел должно быть вещественным. Пример: $100/8 \rightarrow 12$, а вот $100/8.0 \rightarrow 12.5$

Таблица 1.1: Операции с числами

Операция	Описание
$x//y$	Целочисленное деление (результат — целое число). Если оба числа в операции вещественные, получается вещественное число с дробной частью, равной нулю. Пример: $100//8 \rightarrow 12$ $101.8//12.5 \rightarrow 8.0$ (для сравнения $101.8/12.5 \rightarrow 8.1440000000000001$)
$x\%y$	Остаток от целочисленного деления x на y Пример: $10\%4 \rightarrow 2$
$x**y$	Возведение в степень (x в степени y). Работает и для вещественных чисел. Примеры: $2**3 \rightarrow 8$ $2.3*(-3.5) \rightarrow 0.05419417057580235$
$-x$	Смена знака числа

Кроме того, в Python для операций с числами используются функции `abs()` (вычисление абсолютного значения — модуля, `abs(-3) → 3`), `pow()` (возведение в степень, `pow(2,3) → 8`), `divmod()` (вычисление результата целочисленного деления и остатка, `divmod(17,5) → (3,2)`) и `round()` (округление, `round(100.0/6) → 17.0`). Эти функции являются «встроенными», что означает, что для их использования нет необходимости подключать дополнительные модули. Все прочие функции для работы с числами (математические), такие как вычисление квадратного корня, синуса и пр. требуют подключения модуля `math`.

1.2.1.2 Логические значения

Логические значения в Python представлены двумя величинами — логическими константами `True` (Истина) и `False` (Ложь).

Логические значения получаются в результате логических операций и вычисления логических выражений.

Таблица 1.2: Основные логические операции и выражения

Операция или выражение	Описание
$>$	Условие «больше» (например, проверяем, что $a > b$)
$<$	Условие «меньше» (например, проверяем, что $a < b$)
$==$	Условие равенства (проверяем, что a равно b)
$!=$	Условие неравенства (проверяем, что a не равно b)
not x	Отрицание (условие x не выполняется)
x and y	Логическое «И» (умножение). Чтобы выполнилось условие x and y , необходимо, чтобы одновременно выполнялись условия x и y .
x or y	Логическое «ИЛИ» (сложение). Чтобы выполнилось условие x or y , необходимо, чтобы выполнилось одно из условий.
x in A	Проверка принадлежности элемента x множеству (структуре) A (см. «Структуры данных»).
$a < x < b$	Эквивалентно $(x > a)$ and $(x < b)$

1.2.2 Структуры данных

В Python определены такие структуры данных (составные типы) как последовательности и отображения (называемые также словарями). Словари позволяют устанавливать связи (ассоциации) «ключ—значение» (например, «Фамилия—Адрес»), поэтому с их помощью создаются так называемые ассоциативные массивы. В нашем курсе не будут рассматриваться словари и подробности их применения, а для создания ассоциативных массивов будут использоваться другие структуры данных.

Последовательности, в свою очередь, подразделяются на изменяемые и неизменяемые. Под изменяемостью (изменчивостью) последовательности понимается возможность добавлять или удалять элементы этой последовательности (т.е. изменять количество элементов последовательности).

Для структур данных в Python определены функции (операции) и методы, принципиального различия между которыми нет, а есть различие синтаксическое (в правилах написания). Основные функции и методы для каждого типа структур данных приводятся ниже.

1.2.2.1 Неизменяемые последовательности — строки

Строки (последовательности символов — букв и других значков, которые можно найти на клавиатуре компьютера) могут состоять из символов английского и любого другого алфавита. Для простоты и определённости в строковых значениях переменных в дальнейшем будем использовать только символы английского алфавита. В Python строки и символы нужно заключать в кавычки (одиночные или двойные). Элементы (символы) в строке нумеруются, начиная с нуля. Одиночный символ — буква — является «с точки зрения Python» строкой, состоящей из одного элемента.

Максимально возможное количество символов в строке (длина строки) в Python ограничивается только доступным объёмом памяти. Так что текст любого разумного размера (например, несколько тысяч страниц) может быть записан в одну строку Python.

Числа могут быть преобразованы в строки с помощью функции `str()`. Например, `str(123)` даст строку `'123'`. Если строка является последовательностью знаков-цифр, то она может быть преобразована в целое число в помощью функции `int()` (`int('123')` даст в результате число 123), а в вещественное — с помощью функции `float()` (`float('12.34')` даст в результате число 12.34). Для любого символа можно узнать его номер (код символа) с помощью функции `ord()` (например, `ord('s')` даст результат 115). И наоборот, получить символ по числовому коду можно с помощью функции `chr()` (например `chr(100)` даст результат `'d'`).

Таблица 1.3: Основные операции со строками

Функция или операция	Описание и результат
<code>len(s)</code>	Вычисляется длина строки <code>s</code> как число символов
<code>s1 + s2</code>	Конкатенация. К концу строки <code>s1</code> присоединяется строка <code>s2</code> , в результате получается новая строка, например, <code>'вы' + 'года' → 'выгода'</code>
<code>s * n</code> (или <code>n * s</code>)	<code>n</code> -кратное повторение строки <code>s</code> , в результате получается новая строка, например <code>'кан'*2 → 'канкан'</code>
<code>s[i]</code>	Выбор из <code>s</code> элемента с номером <code>i</code> , нумерация начинается с 0 (первый элемент имеет номер 0). Результатом является символ. Если <code>i < 0</code> , отсчёт идёт с конца (первый символ строки имеет номер 0, последний имеет номер <code>-1</code>). Пример: <code>s = 'дерево'</code> <code>s[2] → 'р'</code> <code>s[-2] → 'в'</code>

Таблица 1.3: Основные операции со строками

Функция или операция	Описание и результат
<code>s[i:j:k]</code>	Срез — подстрока, содержащая символы строки <code>s</code> с номерами от <code>i</code> до <code>j</code> с шагом <code>k</code> (элемент с номером <code>i</code> входит в итоговую подстроку, а элемент с номером <code>j</code> уже не входит). Если <code>k</code> не указан (использован вариант <code>s[i:j]</code>), то символы идут подряд (равносильно <code>s[i:j:1]</code>). Примеры: <code>s='derevo'</code> <code>s[3:5] → 'ev'</code> <code>s[1:5:2] → 'ee'</code>
<code>min(s)</code>	Определяет и выводит (возвращает) символ с наименьшим значением (кодом – номером в кодовой таблице) Пример: <code>s='derevo'</code> <code>min(s) → 'd'</code>
<code>max(s)</code>	Возвращает символ с наибольшим значением (кодом) Пример: <code>s='derevo'</code> <code>max(s) → 'v'</code>

Строки, как объекты Python, обладают методами (т.е. функциями, которые выполняют сами объекты). Основные методы перечислены в следующей таблице. Пусть строка, к которой применяются эти методы, называется `s1`.

Таблица 1.4: Методы строк

Метод	Описание и результат
<code>s1.center(n)</code>	Возвращается строка <code>s1</code> , дополненная пробелами справа и слева до ширины в <code>n</code> символов. Исходная строка не изменяется. Если $n \leq \text{len}(s1)$, пробелы не добавляются. Пример: <code>s1='Zoom-Zoom'</code> <code>s1.center(15) → '___Zoom-Zoom___'</code>

Таблица 1.4: Методы строк

Метод	Описание и результат
<code>s1.ljust(n)</code>	<p>Строка <code>s1</code> выравнивается по левому краю (дополняется пробелами справа) в пространстве шириной n символов. Если $n < \text{len}(s1)$, пробелы не добавляются.</p> <p>Пример: <code>s1='Zoom-Zoom'</code> <code>s1.ljust(15) → 'Zoom-Zoom '</code></p>
<code>s1.rjust(n)</code>	<p>Строка <code>s1</code> выравнивается по правому краю (дополняется пробелами слева) в пространстве шириной n символов. Если $n < \text{len}(s1)$, пробелы не добавляются.</p> <p>Пример: <code>s1='Zoom-Zoom'</code> <code>s1.rjust(15) → ' Zoom-Zoom'</code></p>
<code>s1.count(s[i, j])</code>	<p>Определяется количество вхождений подстроки <code>s</code> в строку <code>s1</code>. Результатом является число. Можно указать позицию начала поиска i и окончания поиска j (по тем же правилам, что и начало и конец среза).</p> <p>Примеры: <code>s1='abrakadabra'</code> <code>s1.count('ab') → 2</code> <code>s1.count('ab',1) → 1</code> <code>s1.count('ab',1,-3) → 0</code>, потому что <code>s1[1:-3] → 'brakada'</code></p>
<code>s1.find(s[i, j])</code>	<p>Определяется позиция первого (считая слева) вхождения подстроки <code>s</code> в строку <code>s1</code>. Результатом является число. Необязательные аргументы i и j определяют начало и конец области поиска (как в предыдущем случае).</p> <p>Пример: <code>s1='abrakadabra'</code> <code>s1.find('br') → 1</code></p>

Таблица 1.4: Методы строк

Метод	Описание и результат
<code>s1.rfind(s[, i, j])</code>	<p>Определяется позиция последнего (считая слева) вхождения подстроки <code>s</code> в строку <code>s1</code>. Результатом является число. Необязательные аргументы <code>i</code> и <code>j</code> определяют начало и конец области поиска (как в предыдущем случае).</p> <p>Пример: <code>s1='abrakadabra'</code> <code>s1.rfind('br') → 8</code></p>
<code>s1.strip()</code>	<p>Создаётся копия строки, в которой удалены пробелы в начале и в конце (если они есть или образовались в результате каких-то операций).</p> <p>Пример: <code>s1=' breKeKeKeKs '</code> <code>s2=s1.strip()</code> <code>s2 → 'breKeKeKeKs'</code></p>
<code>s1.lstrip()</code>	<p>Создаётся копия строки, в которой удалены пробелы в начале (если они есть или образовались в результате каких-то операций).</p> <p>Пример: <code>s1=' breKeKeKeKs '</code> <code>s2=s1.lstrip()</code> <code>s2 → 'breKeKeKeKs'</code></p>
<code>s1.rstrip()</code>	<p>Создаётся копия строки, в которой удалены пробелы в конце (если они есть или образовались в результате каких-то операций).</p> <p>Пример: <code>s1=' breKeKeKeKs '</code> <code>s2=s1.rstrip()</code> <code>s2 → 'breKeKeKeKs'</code></p>
<code>s1.replace(s2, s3[, n])</code>	<p>Создаётся новая строка, в которой фрагмент (подстрока) <code>s2</code> исходной строки заменяется на фрагмент <code>s3</code>. Необязательный аргумент <code>n</code> указывает количество замен (если требуется заменить не все фрагменты).</p> <p>Пример: <code>s1='breKeKeKeKs'</code> <code>ss=s1.replace('Ke', 'XoXo', 2)</code> <code>ss → 'breXoXoXoXoKs'</code></p>

Таблица 1.4: Методы строк

Метод	Описание и результат
<code>s1.capitalize()</code>	Создаётся новая строка, в которой первая буква исходной строки становится заглавной (прописной), а все остальные становятся маленькими (строчными). Пример: <code>s1='breKeKeKeKs'</code> <code>s2=s1.capitalize()</code> <code>s2 → 'Brekekekeks'</code>
<code>s1.swapcase()</code>	Создаётся новая строка, в которой прописные буквы исходной строки заменяются на строчные и наоборот. Пример: <code>s1='breKeKeKeKs'</code> <code>s2=s1.swapcase()</code> <code>s2 → 'BREkEkEkEkS'</code>
<code>s1.upper()</code>	Создаётся новая строка, в которой все буквы исходной строки становятся заглавными (прописными). Пример: <code>s1='breKeKeKeKs'</code> <code>s2=s1.upper()</code> <code>s2 → 'BREKEKEKEKS'</code>
<code>s1.lower()</code>	Создаётся новая строка, в которой все буквы исходной строки становятся маленькими (строчными). Пример: <code>s1='breKeKeKeKs'</code> <code>s2=s1.lower()</code> <code>s2 → 'brekekekeks'</code>

1.2.2.2 Неизменяемые последовательности — кортежи

Кортеж в Python — это упорядоченный набор объектов, в который могут одновременно входить объекты разных типов (числа, строки и другие структуры, в том числе и кортежи). В дальнейшем эти объекты будем называть элементами кортежа.

Кортеж задаётся перечислением его элементов в круглых скобках через запятую, например

```
t=(12, 'b', 34.6, 'derevo')
```

С использованием допустимой в Python цепочки присваиваний можно элементам кортежа сразу сопоставить какие-нибудь переменные:

```
t=(x, s1, y, s2)=(12, 'b', 34.6, 'derevo')
```

В этом случае элемент кортежа и соответствующая переменная будут указывать на одни и те же значения, т.е. значение `t[0]` будет равно значению `x`, а `t[3]`, соответственно, `s2`.

Однако эти переменные могут изменяться независимо от элементов кортежа. Присвоение нового значения переменной `s1` никак не влияет на элемент `t[1]`. А вот для элементов кортежа значения изменить уже нельзя, поскольку для Python кортеж относится к неизменяемым последовательностям.

Кортеж может быть пустым (для его определения нужно написать `t=()`), а может содержать только один элемент (например, `t=('domik',)`). Для кортежа из одного элемента обязательно добавлять запятую после имени или значения этого элемента.

Кортежи могут получаться в результате работы функций Python, например, уже упоминавшаяся функция `divmod()` возвращает кортеж из двух элементов.

Кортежи могут использоваться для хранения характеристик каких-нибудь предметов, существ или явлений, если эти предметы, существа или явления характеризуются фиксированным набором свойств. Например, в виде кортежа можно записать фамилию ученика и его оценки за полугодие.

Поскольку кортежи являются неизменяемыми последовательностями, операции с кортежами не меняют исходные кортежи.

Таблица 1.5: Основные операции с кортежами

Функция или операция	Описание и результат
<code>len(t)</code>	Определяется количество элементов кортежа (результатом является число) <code>t</code>
<code>t1 + t2</code>	Объединение кортежей. Получается новый кортеж, в котором после элементов кортежа <code>t1</code> находятся элементы кортежа <code>t2</code> . Пример: <code>t1=(1,2,3)</code> <code>t2=('raz', 'dva')</code> <code>t3=t1+t2</code> <code>t3 → (1, 2, 3, 'raz', 'dva')</code>
<code>t * n</code> или <code>(n * t)</code>	<code>n</code> -кратное повторение кортежа <code>t</code> Пример: <code>t2=('raz', 'dva')</code> <code>t2*3 → ('raz','dva', 'raz', 'dva', 'raz', 'dva')</code>

Таблица 1.5: Основные операции с кортежами

Функция или операция	Описание и результат
<code>t[i]</code>	Выбор из <code>t</code> элемента с номером <code>i</code> , нумерация начинается с 0 (первый элемент имеет номер 0) Если <code>i < 0</code> , отсчёт идёт с конца (первый элемент кортежа имеет номер 0, последний имеет номер <code>-1</code>). Пример: <code>t3 = (1, 2, 3, 'raz', 'dva')</code> <code>t3[2] → 3</code> <code>t3[-2] → 'raz'</code>
<code>t[i:j:k]</code>	Срез — кортеж, содержащий элементы кортежа <code>t</code> с номерами от <code>i</code> до <code>j</code> с шагом <code>k</code> (элемент с номером <code>i</code> входит в итоговый кортеж, а элемент с номером <code>j</code> уже не входит). Если <code>k</code> не указан (использован вариант <code>t[i:j]</code>), то элементы идут подряд (равносильно <code>t[i:j:1]</code>). Пример: <code>t3 = (1, 2, 3, 'raz', 'dva')</code> <code>t3[1:4] → (2, 3, 'raz')</code>
<code>min(t)</code>	Определяется элемент с наименьшим значением в соответствии с алфавитным («словарным») порядком. Пример: <code>t3 = (1, 2, 3, 'raz', 'dva')</code> <code>min(t3) → 1</code>
<code>max(t)</code>	Определяется элемент с наибольшим значением в соответствии с алфавитным («словарным») порядком. Пример: <code>t3 = (1, 2, 3, 'raz', 'dva')</code> <code>max(t3) → 'raz'</code>

Важно понимать, что «словарный» порядок — сначала числа по возрастанию, затем строки, начинающиеся на цифры в порядке их возрастания, затем строки, начинающиеся на прописные буквы в алфавитном порядке, а затем строки, начинающиеся на строчные буквы также в алфавитном порядке – всегда используется в вычительной технике при сортировке имён объектов. Строку можно преобразовать в кортеж с помощью функции `tuple()`, например:

```
s='amamam'
t=tuple(s)
t → ('a', 'm', 'a', 'm', 'a', 'm')
```

При работе с кортежами заменить значение элемента кортежа нельзя. Если при написании программы возникает такая необходимость, это может свидетельствовать о том, что кортеж является неподходящей структурой данных для решения задачи. Возможно, в этой ситуации необходимо использовать изменяемые последовательности (списки, см. ниже).

1.2.2.3 Изменяемые последовательности — списки

Список в Python — это упорядоченный набор объектов, в список могут одновременно входить объекты разных типов (числа, строки и другие структуры, в частности, списки и кортежи). Объекты, входящие в список, будем в дальнейшем называть элементами списка.

Самый наглядный способ создания списка — перечислить его элементы в квадратных скобках через запятую, например:

```
lst = [12, 'b', 34.6, 'derevo']
```

В дальнейшем в именах списков всегда будем использовать сочетание `lst` (от слова «list», т.е. «список»).

С использованием допустимой в Python цепочки присваиваний можно элементам списка сразу сопоставить какие-нибудь переменные:

```
lst = [x, s1, y, s2] = [12, 'b', 34.6, 'derevo']
```

В этом случае элемент списка и соответствующая переменная будут указывать на одни и те же значения, т.е. значение `lst[0]` будет равно значению `x`, а `lst[3]` соответственно, `s2`.

Однако эти переменные могут изменяться независимо от элементов списка. Присвоение нового значения переменной `s1` никак не влияет на элемент `lst[1]`. В отличие от кортежа, значения элементов списка можно изменять, добавлять элементы в список и удалять их.

Список может быть пустым (создать его можно так: `lst = []`), а может содержать только один элемент (например, `lst = ['domik']`).

Списки являются очень полезными структурами данных в Python, и с использованием списков, их методов и операций с ними можно эффективно решать самые разнообразные задачи.

Таблица 1.6: Основные операции со списками

Функция или операция	Описание и результат
<code>len(lst)</code>	Определяется количество элементов списка <code>lst</code> . Результат — число.

Таблица 1.6: Основные операции со списками

Функция или операция	Описание и результат
<code>lst1 + lst2</code>	Объединение списков. Получается новый список, в котором после элементов списка <code>lst1</code> находятся элементы списка <code>lst2</code> . Пример: <code>lst1 = [1, 2, 3]</code> <code>lst2 = ['raz', 'dva']</code> <code>lst3 = lst1 + lst2</code> <code>lst3 → [1, 2, 3, 'raz', 'dva']</code>
<code>lst * n</code> (или <code>n * lst</code>)	n-кратное повторение списка <code>lst</code> . Результат — новый список. Пример: <code>lst2 = ['raz', 'dva']</code> <code>lst2 * 3 → ['raz', 'dva', 'raz', 'dva', 'raz', 'dva']</code>
<code>lst[i]</code>	Выбор из <code>lst</code> элемента с номером <i>i</i> , нумерация начинается с 0 (первый элемент имеет номер 0). Если <i>i</i> < 0, отсчёт идёт с конца (последний элемент списка имеет номер -1). Пример: <code>lst3 = [1, 2, 3, 'raz', 'dva']</code> <code>lst3[2] → 3</code> <code>lst3[-2] → 'raz'</code>
<code>lst[i:j:k]</code>	Срез — список, содержащий элементы списка <code>lst</code> с номерами от <i>i</i> до <i>j</i> с шагом <i>k</i> (элемент с номером <i>i</i> входит в итоговый список, а элемент с номером <i>j</i> уже не входит). Если <i>k</i> не указан (использован вариант <code>lst[i:j]</code>), то символы идут подряд (равносильно <code>lst[i:j:1]</code>). Пример: <code>lst3 = [1, 2, 3, 'raz', 'dva']</code> <code>lst3[1:4] → [2, 3, 'raz']</code>
<code>min(lst)</code>	Определяется элемент с наименьшим значением в соответствии с алфавитным («словарным») порядком. Пример: <code>lst3 = [1, 2, 3, 'raz', 'dva']</code> <code>min(lst3) → 1</code>

Таблица 1.6: Основные операции со списками

Функция или операция	Описание и результат
<code>max(lst)</code>	<p>Определяется элемент с наибольшим значением в соответствии с алфавитным («словарным») порядком.</p> <p>Пример: <code>lst3 = [1, 2, 3, 'raz', 'dva']</code> <code>max(lst3) → 'raz'</code></p>
<code>lst[i]=x</code>	<p>Замена элемента списка с номером i на значение x. Если x является списком, то на место элемента списка будет вставлен список. При этом новый список не создаётся.</p> <p>Примеры: <code>lst3=[1, 2, 3, 'raz', 'dva']</code> <code>lst3[2]='tri'</code> <code>lst3 → [1, 2, 'tri', 'raz', 'dva']</code> <code>lst3[2]=[7,8]</code> <code>lst3 → [1, 2, [7, 8], 'raz', 'dva']</code></p>
<code>del lst[i]</code>	<p>Удаление из списка элемента с номером i. Новый список не создаётся.</p> <p>Пример: <code>lst3=[1, 2, [7, 8], 'raz', 'dva']</code> <code>del lst3[2]</code> <code>lst3 → [1, 2, 'raz', 'dva']</code></p>
<code>lst[i:j]=x</code>	<p>Замена среза списка <code>lst</code> на элемент или список x (несколько элементов заменяются на x).</p> <p>Примеры: <code>lst3=[1, 2, 3, 'raz', 'dva']</code> <code>lst3[2:4]='tri'</code> <code>lst3 → [1, 2, 't', 'r', 'i', 'dva']</code> <code>lst3[2:4]='a'</code> <code>lst3 → [1, 2, 'a', 'i', 'dva']</code></p> <p>Обратите внимание, что строка интерпретируется как список!</p>
<code>del lst[i:j]</code>	<p>Удаление элементов, входящих в указанный срез («вырезание среза»).</p> <p>Пример: <code>lst3=[1, 2, 'a', 'i', 'dva']</code> <code>del lst3[2:4]</code> <code>lst3 → [1, 2, 'dva']</code></p>

Важно понимать, что при определении значений минимального и максимального элементов списка также используется «словарный» порядок — сначала идут числа по возрастанию, затем строки, начинающиеся на цифры в порядке их возрастания, затем строки, начинающиеся на прописные буквы в алфавитном порядке, а затем строки, начинающиеся на строчные буквы также в алфавитном порядке.

Списки в Python, как и строки, являются объектами, поэтому для списков существуют методы.

Таблица 1.7: Основные методы списков

Метод	Описание и результат
<code>lst.append(x)</code>	Добавление элемента <code>x</code> в конец списка <code>lst</code> . <code>x</code> не может быть списком. Создания нового списка не происходит. Пример: <code>lst=['raz','dva','tri',1,2]</code> <code>lst.append(3)</code> <code>lst → ['raz','dva','tri',1,2,3]</code>
<code>lst.extend(t)</code>	Добавление кортежа или списка <code>t</code> в конец списка <code>lst</code> (похоже на объединение списков, но создания нового списка не происходит). Пример: <code>lst1=[1,2,3]</code> <code>lst2=['raz','dva']</code> <code>lst1.extend(lst2)</code> <code>lst1 → [1,2,3,'raz','dva']</code>
<code>lst.count(x)</code>	Определение количества элементов, равных <code>x</code> , в списке <code>lst</code> . Результат является числом. Пример: <code>lst=[1,2,3,'raz','dva','raz','dva']</code> <code>lst.count('raz') → 2</code>
<code>lst.index(x)</code>	Определение первой слева позиции элемента <code>x</code> в списке <code>lst</code> . Если такого элемента нет, возникает сообщение об ошибке. Пример: <code>lst=[1,2,3,'raz','dva','raz','dva']</code> <code>lst.index('dva') → 4</code>

Таблица 1.7: Основные методы списков

Метод	Описание и результат
<code>lst.remove(x)</code>	Удаление элемента <code>x</code> в списке <code>lst</code> в первой слева позиции. Если такого элемента нет, возникает сообщение об ошибке. Пример: <code>lst=[1,2,3,'raz','dva','raz','dva']</code> <code>lst.remove('dva')</code> <code>lst → [1,2,3,'raz','raz','dva']</code>
<code>lst.pop(i)</code>	Удаление элемента с номером <code>i</code> из списка <code>lst</code> . При этом выдаётся значение этого элемента («извлечение» элемента из списка). Если номер не указан, удаляется последний элемент. Новый список не создаётся. Примеры: <code>lst=[1,2,3,'raz','raz','dva']</code> <code>\ lstinline lst.pop(3) → 'raz' </code> <code>lst → [1,2,3,'raz','dva']</code> <code>\ lstinline lst.pop() → 'dva' </code> <code>lst → [1,2,3,'raz']</code>
<code>lst.insert(i,x)</code>	Вставка элемента или списка <code>x</code> в позицию <code>i</code> списка <code>lst</code> . Если <code>i ≥ 0</code> , вставка идёт в начало списка. Если <code>i > len(lst)</code> , вставка идёт в конец списка. Новый список не создаётся. Пример: <code>lst=[1,2,3,'raz']</code> <code>lst.insert(3,'tri')</code> <code>lst → [1,2,3,'tri','raz']</code>
<code>lst.sort()</code>	Сортировка списка по возрастанию (в алфавитном порядке). Новый список не создаётся. Пример: <code>lst=[1,2,3,'tri','raz']</code> <code>lst.sort()</code> <code>lst → [1,2,3,'raz','tri']</code>
<code>lst.reverse()</code>	Замена порядка следования элементов на обратный. Новый список не создаётся. Пример: <code>lst=[1,2,3,'raz','tri']</code> <code>lst.reverse()</code> <code>lst → ['tri','raz',3,2,1]</code>

Кроме перечисленных операций и методов, списки могут обрабатываться совместно (по номерам соответствующих элементов). Для этого в Python используются функции `zip()` и `map()`.

Функция `zip()` позволяет получить из элементов различных списков список кортежей, состоящих из соответствующих элементов списков. Аргументами функции `zip()` являются два или более списков, а результатом — список кортежей, составленных из элементов исходных списков с одинаковыми номерами (первый кортеж составляется из элементов с номером 0, второй — из элементов с номером 1 и т.д.)

Пример:

```
lst1=[1,2,3,4]
lst2=['tri','dva','raz']
lst=zip(lst1,lst2)
lst → [(1, 'tri'), (2, 'dva'), (3, 'raz')]
```

Количество элементов в итоговом списке равно количеству элементов в самом коротком исходном списке. «Лишние» элементы других списков игнорируются.

Функцию `zip()` можно применять и к кортежам, а также «смешивать» в её аргументах списки и кортежи.

Функция `map()` используется для применения одной и той же операции к элементам одного или нескольких списков или кортежей. Если списков (кортежей) несколько, они должны быть одинаковой длины (иметь одинаковое количество элементов). При использовании `map()` чаще всего применяются так называемые `lambda`-функции, т.е. безымянные функции, действующие только на время конкретной операции `map()`.

При создании `lambda`-функции указывается ключевое слово **`lambda`**, затем пишутся переменные, для которых эта функция определяется и операции с этими переменными (что функция делает с переменными).

После описания `lambda`-функции, которая является первым аргументом функции `map()` пишутся остальные аргументы — имена списков (кортежей), с которыми надо выполнить операцию.

Примеры:

```
lst1=[1,2,3,4]
lst=map(lambda x: x*2,lst1)
lst → [2, 4, 6, 8]
t1=(1,2,3)
t2=(5.0,6.0,7.0)
t=map(lambda x,y: x/y,t1,t2)
t → [0.20000000000000001, 0.33333333333333331,
0.42857142857142855]
```

В случае, если в функции `map()` в качестве первого аргумента используется специальная функция `None`, результат равносителен использованию функции `zip()`. Другими словами, `map(None, lst1, lst2)` равносильно `zip(lst1, lst2)`.

Для списков и кортежей, состоящих только из чисел, возможно применение функции `sum()`, которая вычисляет сумму элементов списка (кортежа).

Примеры:

```
lst1=[1,2,3,4]
sum(lst1) → 10
t1=(1,2,3)
sum(t1) → 6
```

Для преобразования строки или кортежа в список используется функция `list()`.

Примеры:

```
s='amamam'
lst=list(s)
lst → ['a', 'm', 'a', 'm', 'a', 'm']
t=(5, 12, -3, 7)
lst2=list(t)
lst2 → [5, 12, -3, 7]
```

Соответственно, с помощью функции `tuple()` список может быть преобразован в кортеж.

Также для взаимного преобразования строк и списков используются методы строк `split()` и `join()`. Метод `split()` делит строку по заданному символу-разделителю и создаёт список из фрагментов строки.

Пример:

```
s='mama_myla_ramu'
lst=s.split('_') # символ-разделитель — пробел
lst → ['mama', 'myla', 'ramu']
```

Метод `join()` формирует строку из элементов списка, поставив между ними заданную строку (соединяет элементы списка с помощью заданной строки).

Пример:

```
lst=['1', '2', '3']
s='nea'.join(lst)
s → '1nea2nea3'
```

Функция `range()` создаёт список как числовую арифметическую прогрессию. Полный вариант её использования:

```
range(x0, x1, d)
```

При этом создаётся список из чисел в полуоткрытом интервале $[x_0, x_1)$ с шагом d , например,

```
range(0,15,3) → [0, 3, 6, 9, 12]
```

Минимальный вариант

```
range(n)
```

создаёт список чисел от 0 до $n - 1$ с шагом 1.

Промежуточный вариант

```
range(k, n)
```

создаёт список чисел от k до $n - 1$ с шагом 1.

Для списков, созданных с помощью функции `range()`, часто используются проверки принадлежности величины x списку `range()` (условие `x in range(a,b,d)`) или непринадлежности (условие `x not in range(a,b,d)`). Такие условия встречаются при организации циклов с переменной (см. раздел 2.3).

Применение функции `sum()` для списков, полученных с помощью `range()`, даёт сумму прогрессии.

Примеры:

```
sum(range(10)) → 45  
sum(range(0,15,3)) → 30
```

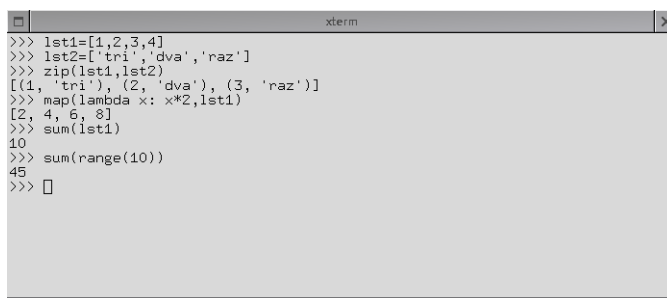
1.3 Средства программирования на Python

Python имеет возможность работы в режиме интерпретатора, в котором команды и операции выполняются сразу после их ввода. Вызов интерпретатора Python осуществляется набором команды `python` в командной строке. Пример работы в сеансе интерпретатора (интерактивной оболочки) показан на рис. 1.1. Для выхода из этого режима используется комбинация клавиш `<CTRL>+<D>`.

В интерактивной оболочке команды и операции вводятся с клавиатуры после знака приглашения интерпретатора `>>>`. Ввод каждой операции завершается нажатием на клавишу `<ENTER>`, после чего Python выполняет эту операцию и выдаёт результат или сообщение об ошибке. После присваивания результата операции какой-нибудь переменной никакой результат не выдаётся, а чтобы его увидеть, нужно набрать имя переменной и нажать `<ENTER>`.

Однако в интерактивной оболочке неудобно работать с файлами программ. Кроме того, полезно видеть текст программы одновременно с результатами её выполнения. Такие функции (и часто многие другие) обеспечивают интегрированные среды разработки (IDE — Integrated Development Environment). Одним из достоинств IDE является подсветка синтаксиса — команды, строки, числа и другие элементы программ и данных выделяются цветом или начертанием шрифта.

Самая простая IDE для Python называется IDLE (рис. 1.2). В этой среде можно редактировать тексты программ в окне редактора и запускать их на выполнение. Результаты выполнения отображаются в окне выполнения, которое одновременно является окном интерактивной оболочки Python (т.е. в этом окне также можно выполнять команды).



```
>>> lst1=[1,2,3,4]
>>> lst2=['tri','dva','raz']
>>> zip(lst1,lst2)
[(1, 'tri'), (2, 'dva'), (3, 'raz')]
>>> map(lambda x: x*2,lst1)
[2, 4, 6, 8]
>>> sum(lst1)
10
>>> sum(range(10))
45
>>> □
```

Рис. 1.1: Сеанс в интерпретаторе Python



Рис. 1.2: Интегрированная среда разработки IDLE

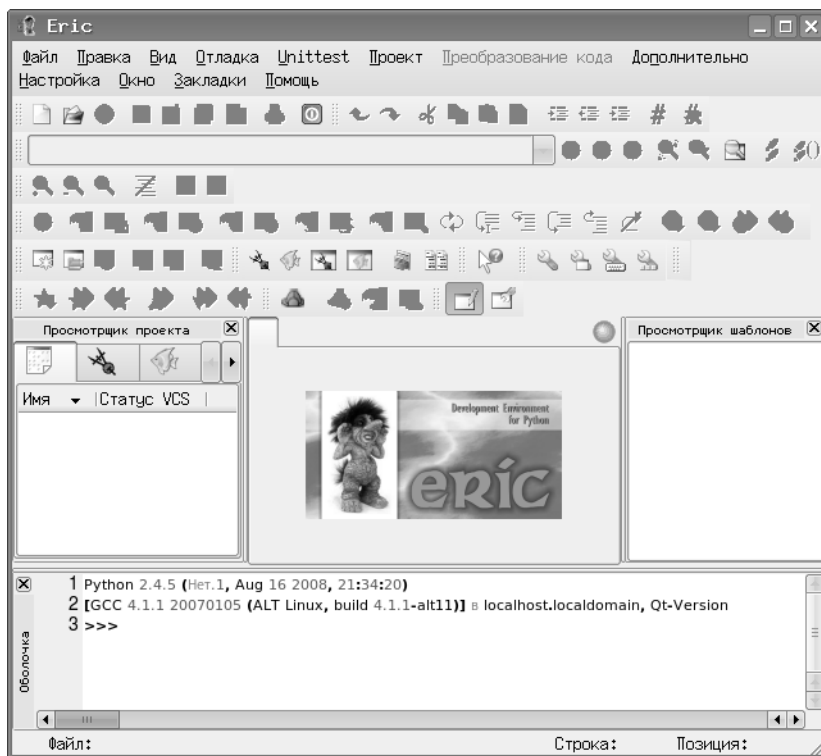


Рис. 1.3: IDE Eric

Пункт «Options» главного меню IDLE позволяет выбрать, какое окно будет открываться при запуске программы — окно редактора или окно оболочки. Для выполнения программы, набранной в окне редактора, нужно нажать <F5>. Если файл с текстом программы не сохранён, IDLE выдаст соответствующее сообщение и предложит сохранить файл. Если окно оболочки при этом отсутствует, оно автоматически откроется и покажет результаты выполнения программы.

Недостатком IDLE является «бедный» и не локализованный (только на английском) интерфейс, а достоинством то, что реализации IDLE существуют для всех распространённых операционных систем.

Также специально для Python разработана IDE Eric (рис. 1.3), которая обладает большим количеством настроек и возможностей отладки программ и больших программных проектов. Однако эту среду разработки (точнее, её внешний вид) можно существенно упростить и сделать её более понятной для начинающих (рис. 1.4).

Важно обратить внимание, что в Eric также имеется окно редактора и окно выполнения (окно интерактивной оболочки).

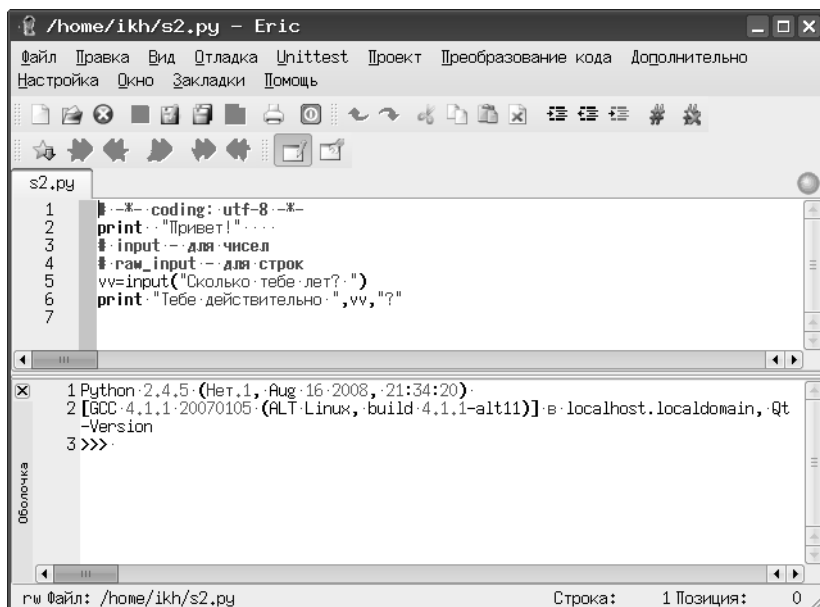


Рис. 1.4: Упрощённый вариант IDE Eric

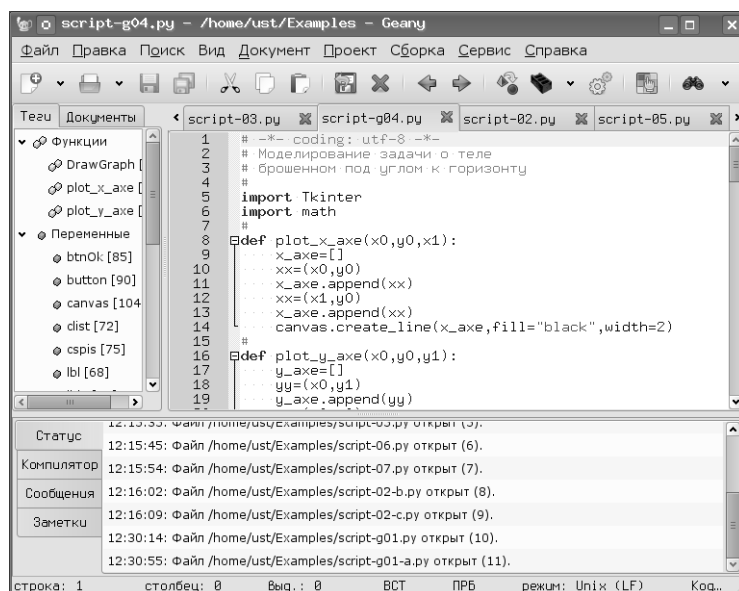


Рис. 1.5: Основное окно IDE Geany

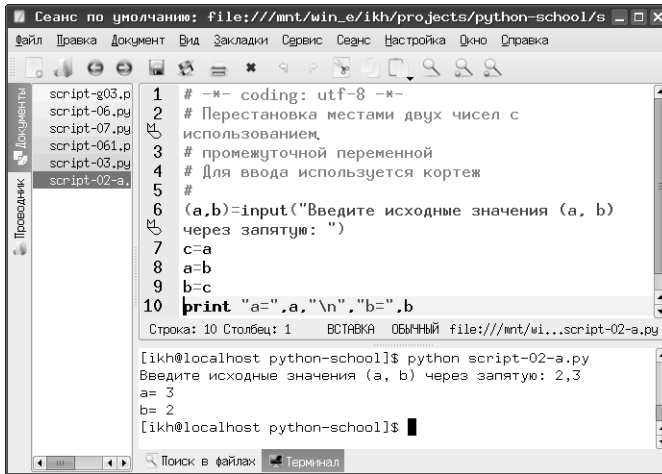


Рис. 1.6: Текст программы и процесс её выполнения в редакторе Kate

Интерес представляет также кросс-платформенная IDE Geany (рис. 1.5), в которой, кроме Python, можно работать со многими другими языками программирования.

Кроме того, для создания и выполнения программ на Python (как и многих других языках программирования) можно использовать текстовые редакторы для программистов, в частности, редактор Kate, входящий в состав интегрированной среды KDE (рис. 1.6).

Для запуска программы используется команда `python имя_файла.py` в окне терминала Kate.

1.4 Ввод и вывод в Python

1.4.1 Диалоговый режим

При работе с интерактивными оболочками или в процессе организации взаимодействия программы с пользователем («диалога») для ввода чисел и, соответственно, определения значений переменных будем использовать функции `input()` и `raw_input()`. В качестве аргумента этих функции рекомендуется использовать строку-подсказку (приглашение для ввода), в которой кратко описывается, какие данные и как необходимо сообщить программе. Далее будут показаны примеры использования этих функций.

Если в строке-подсказке используются символы кириллицы (русские буквы), то нужно предварительно указать кодировочную таблицу (см. раздел 1.5). Строка-подсказка может быть в двойных или в одиночных кавычках. Для выполнения операций `input()` или `raw_input()` интерпретатор останавливает програм-

му и после строки-подсказки требуется ввести требуемое значение переменной и нажать <ENTER>. Если строка подсказки отсутствует, будет показан курсор оболочки в пустой строке окна выполнения. Если требуется ввести несколько значений, их нужно вводить через запятую и нажимать <ENTER> только после последнего введённого значения.

Рассмотрим примеры ввода данных и вывода результатов команд в сеансе интерактивной оболочки.

В последующих примерах обозначение >>> используется в качестве приглашения интерактивной оболочки, обычный шрифт показывает то, что следует набирать на клавиатуре, **полужирным шрифтом** показан вывод интерпретатора (то, что пишет Python), а символ # означает комментарий (пояснение).

При использовании функции input() числовые значения пишутся как обычно, а строковые нужно писать в кавычках (двойных или одинарных).

Примеры:

```
>>> a=input() <ENTER>           # простейший вариант
12 <ENTER>
>>> a <ENTER>                   # проверяем значение переменной a
12
>>>

>>> a=input('Введите значение a: ') <ENTER>    # вариант с
подсказкой
Введите значение a: 12 <ENTER>    # подсказка и ввод
>>>

>>> a,b,c=input('Введите длины сторон треугольника через запятую: ')
<ENTER>           # вариант для нескольких чисел
Введите длины сторон треугольника через запятую: 3,4,5 <ENTER>
>>> b <ENTER>           # проверяем значение переменной b
4
>>>

>>> t=(a,b,c,d)=input('Введите элементы: ') <ENTER>    #
формирование кортежа со строковыми элементами
>>> 5,'bob',-3,'dno' <ENTER>
>>> t[1] <ENTER>           # проверяем значение элемента
'bob'
>>>t <ENTER>
(5, 'bob', -3, 'dno')
>>>
```

Список таким образом ввести не получается (при вводе нескольких значений функция input() возвращает кортеж), но список можно получить с помощью функции list().

Для ввода только строковых значений в Python используется функция `raw_input()`. Её особенности во многом совпадают с функцией `input()`. Есть одна деталь — строковые значения при их вводе не нужно заключать в кавычки. Если с помощью `raw_input()` вводить числа, они преобразуются в строки.

Примеры:

```
>>> name=raw_input('Как_тебя_зовут?_') <ENTER>
Как тебя зовут? Вася <ENTER>
>>> name <ENTER>
'Вася'
>>> age=raw_input('Сколько_тебе_лет?_') <ENTER>
Сколько тебе лет? 12 <ENTER>
>>> age <ENTER>
'12'
>>>
```

Для вывода результатов работы используется инструкция **print**, которая не является функцией. Использование инструкции (команды) **print** позволяет производить вычисления «на лету» и выводить одновременно (одним оператором) строки и числа.

Примеры:

```
>>> print '==stroka==' <ENTER>           # вывод текста
==-stroka==
>>>
>>> t=(a,b,c,d)=input('Введите_элементы:_') <ENTER>
5,'bob',-3,'dno' <ENTER>
>>> print t <ENTER>
(5, 'bob', -3, 'dno')
>>>
>>> print 'Получились_значения', t <ENTER>   # Вывод с
      пояснением
Получились значения (5, 'bob', -3, 'dno')
>>>
```

При выводе нескольких значений через запятую **print** автоматически добавляет пробелы между значениями переменных.

```
>>> t=(a,b,c,d)=input('Введите_элементы:_') <ENTER>
5,'bob',-3,'dno' <ENTER>
>>> t1=(1,2,3) <ENTER>
>>> print 'Итоговый_кортеж', t+t1 <ENTER>   # Пояснение и
      действие
Итоговый кортеж (5, 'bob', -3, 'dno', 1, 2, 3)
>>>
```

1.4.2 Чтение из файла и запись в файл

Будем рассматривать только самый простой вариант — работа с текстовыми файлами в текущем каталоге без указания формата данных.

Для работы с файлом прежде всего нужно создать специальный объект — «дескриптор файла», а потом использовать методы этого дескриптора для чтения и записи данных.

При создании дескриптора нужно указать строку с именем файла и строку с описанием варианта доступа. Вариантов доступа бывает три — 'r' (только чтение), 'w' (запись) и 'a' (дополнение).

Чтение возможно только из существующего файла. Если при открытии на запись или дополнение указано имя несуществующего файла, он будет создан.

Рассмотрим несколько примеров.

Пусть имеется файл с данными с именем 1.dat

```
1 2 3
a b c
```

Создадим дескриптор для чтения данных из этого файла:

```
>>> fd=open('1.dat', 'r') <ENTER>
```

Прочитаем строки из файла:

```
>>> s=fd.read() <ENTER>
>>> s <ENTER>
'1 2 3\na b c'
>>>
```

Как видно, все данные из файла прочитались в одну строку. Поскольку в Python отсутствуют формальные ограничения на длину строки, этот способ годится для любых разумных файлов данных, особенно используемых и учебных целях. Далее с помощью функций и методов строк можно получить значения переменных. Здесь нужно обратить внимание на сочетание '\n'. Это специальная комбинация символов, означающая переход на новую строку (перевод строки — new line). Её можно использовать в своих интересах.

Применение метода `split()` позволяет сформировать список с исходными данными.

```
>>> lst=s.split('\n') <ENTER>
>>> lst <ENTER>
['1 2 3','a b c']
```

Теперь, чтобы получить числа из файла с данными, следует применить метод `split()` к первому элементу списка `lst`. а потом выполнить преобразование типов в зависимости от вида требуемого числа (целое или вещественное).

```
>>> lst2=lst[0].split(' ') <ENTER>
```

```
>>> lst2 <ENTER>
['1','2','3']
>>> b=int(lst2[1]) <ENTER>
>>> b <ENTER>
2
```

Для прекращения работы с файлом («высвобождения дескриптора») используется метод `close()`:

```
>>> fd.close() <ENTER>
```

Есть ещё два полезных метода для чтения данных из файла. Метод `readline()` читает из файла строку, а при повторном использовании — следующую строку:

```
>>> fd=open('1.dat','r') <ENTER>
>>> s01=fd.readline() <ENTER>
>>> s01 <ENTER>
'1 2 3\n'
>>> s01=fd.readline() <ENTER>
'a b c'
>>> fd.close() <ENTER>
```

Метод `readlines()` читает из файла все строки и формирует из них список. Тогда легко узнать количество строк в файле:

```
>>> fd=open('1.dat','r') <ENTER>
>>> lst01=fd.readlines() <ENTER>
>>> lst01 <ENTER>
['1 2 3\n', 'a b c']
>>> len(lst01) <ENTER>
2
>>> fd.close() <ENTER>
```

Далее попытаемся записать что-нибудь в этот файл. Для этого нужно создать дескриптор для записи данных, сформировать строку и использовать метод `write()`, как показано ниже.

```
>>> fd=open('1.dat','w') <ENTER>
>>> s2='c_d_e' <ENTER>
>>> fd.write(s2) <ENTER>
>>> fd.close() <ENTER>
```

В результате содержимое файла заменится на строку `'c_d_e'`, а то, что было в файле раньше, сотрётся. Метод `close()` обеспечивает применение изменений в файле и запись этих изменений на диск.

Теперь рассмотрим случай создания файла и добавления данных в него.

```
>>> fd=open('2.dat','a') <ENTER>
```

```
>>> s1='c_d_e\n' <ENTER>
>>> fd.write(s1) <ENTER>
>>> s2='3_4_5\n' <ENTER>
>>> fd.write(s2) <ENTER>
>>> fd.close() <ENTER>
```

Здесь как раз использована комбинация '\n' для перехода на новую строку. В результате получим файл 2.dat следующего содержания:

```
c d e
3 4 5
```

1.5 Структура программы

Программа на Python представляет из себя последовательность команд для ввода данных, вычислений и других операций с данными и вывода результатов. Простые команды (операторы) принято записывать по одной строке на оператор. В составных операторах (с которыми ещё предстоит познакомиться) большую роль играют пробелы в начале строки (отступы).

Программа создаётся в виде текстового файла в любом текстовом редакторе. Использование интегрированных сред разработки (IDE) обеспечивает подсветку синтаксиса и выделение особенностей структуры программы, а также упрощает поиск ошибок в написании команд. Файл с программой должен иметь «расширение» .py (например, my_program.py).

Первую строку программы необходимо оформить как комментарий, в котором указывается кодировочная таблица («кодировка») данных файла, в противном случае Python не сможет правильно интерпретировать любые символы, встретившиеся в тексте программы и выходящие за пределы основной латиницы (в том числе кириллицу). Указанная в этом комментарии кодировка должна соответствовать действительной кодировке файла.

Пример простейшей программы:

```
# -*- coding: utf-8 -*-
name=raw_input('Как тебя зовут? ')
print 'Здравствуй, ', name, ' !'
```

Для выполнения программы из командной строки следует вызвать интерпретатор Python, указав в качестве аргумента имя файла с программой, например

```
python my_program.py
```

Каждая IDE для Python предлагает свой способ выполнения программы. В IDLE для этого нужно в окне редактора нажать клавишу <F5>, а в Eric — клавиши <F2> и <ENTER>.

1.6 Справочная система и получение информации о Python

Основой справочной системы Python является сам Python и команда `help`, которую можно использовать в интерактивной оболочке.

```
>>> help <ENTER>
Type help() for interactive help, or help(object) for help
  about object.
>>>
```

Для дальнейшего взаимодействия со справочной системой Python требуется некоторое знание английского языка или «помощь друга». Встроенная справка Python существует только на английском языке.

Будет считать, что с английскими особенностями проблем нет, и последуем рекомендации.

```
>>> help() <ENTER>
Welcome to Python 2.4! This is the online help utility.
If this is your first time using Python, you should
  definitely check out
the tutorial on the Internet at
  http://www.python.org/doc/tut/.
Enter the name of any module, keyword, or topic to get help
  on writing
Python programs and using Python modules. To quit this help
  utility
and return to the interpreter, just type "quit".
To get a list of available modules, keywords, or topics,
  type "modules",
"keywords", or "topics". Each module also comes with a
  one-line
summary of what it does; to list the modules whose summaries
  contain a
given word
such as "spam", type "modules_spam".
help>
```

Из результатов работы команды `help()` можно узнать, во-первых, версию Python, во-вторых, адрес сайта с документацией, а в-третьих, получить названия основных разделов справки — модули Python («modules»), ключевые слова Python («keywords») и темы справки («topics»).

После выполнения команды `help()` Python переходит в режим интерактивной справки, соответственно изменяя приглашение оболочки.

Попробуем узнать список ключевых слов, чтобы случайно не использовать их в качестве имён переменных.

```
help> keywords <ENTER>
```

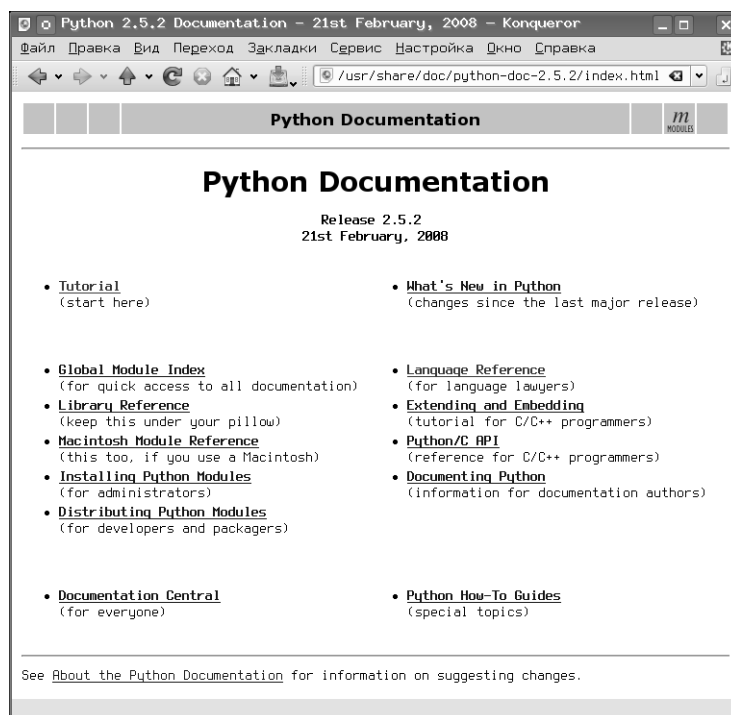


Рис. 1.7: Интерактивная справка по Python в браузере

Here is a list of the Python keywords. Enter any keyword to get more

```
help.
and          else          import        raise
assert       except         in           return
break        exec           is           try
class        finally      lambda       while
continue     for            not          yield
def          from           or
del          global        pass
elif        if            print
help> quit <ENTER>
>>>
```

Для выхода из интерактивной справки используется команда quit.

Справку по Python в виде гипертекста (рис. 1.7) и тоже на английском языке можно получить, открыв в любом браузере файл /usr/share/doc/

`python-doc-x.y.z/index.html`, где `x.y.z` — версия Python (например, `/usr/share/doc/python-doc-2.5.2/index.html`).

Основные учебные материалы, электронные и печатные книги по Python перечислены в разделе «Литература».

1.7 Контрольные вопросы

1. Почему операция вида `a<b=c` недопустима, а операция вида `a<b==c` — допустима?
2. Чем отличаются результаты операций «/» и «//» для целых чисел? А для вещественных чисел?
3. Какая структура является результатом работы функции `divmod()`?
4. Какие ограничения на длину строки установлены в Python?
5. Пусть имеются две строки `s1` и `s2`. Есть ли разница в результатах выполнения команды «**print** `s1+s2`» и команды «**print** `s1,s2`»?
6. Пусть имеется два кортежа `t1` и `t2`. Есть ли разница в результатах выполнения команды «**print** `t1+t2`» и команды «**print** `t1,t2`»?
7. Назовите минимум три отличия списка от кортежа.
8. Пусть имеется строка `s='madagaskar'`. Какая строка будет результатом операции среза `s[1:7:2]`?
9. Опишите последовательность действий, с помощью которых можно получить из файла, в котором записаны четыре вещественных числа, эти числа в виде значений переменных.
10. (Трудный) Опишите способ получения геометрической прогрессии со знаменателем q и вычисления её суммы.
11. (Трудный) Опишите два способа изменить порядок элементов кортежа из четырёх элементов на противоположный.

Глава 2

Основные алгоритмы и их реализация на Python

При разборе задач в этой части будем обращать внимание на постановку задачи (что именно нужно сделать) и собственно алгоритм, который будет описываться как блок-схемой, так и на «псевдоязыке» программирования (подобие «школьного алгоритмического языка»). И только после этого можно приступить к написанию программы на Python с учётом всех его особенностей и возможностей, которые были описаны в предыдущей части.

2.1 Линейные алгоритмы. Операции с числами и строками

Линейный алгоритм — алгоритм, в котором вычисления выполняются строго последовательно. Типичная блок-схема линейного алгоритма показана на рис. 2.1.

Далее рассмотрим типичные задачи с линейной структурой алгоритма.

Задача 1. Дано два числа a и b . Сделать так, чтобы их значения поменялись местами.

Постановка задачи: Имеются две переменные с какими-то определёнными значениями. Пусть значение a равно x , а значение b равно y . Требуется, чтобы значение a стало равно y , а значение b стало равно x .

Метод решения (общий): Использовать дополнительную переменную c , в которую временно записать начальное значение переменной a , присвоить переменной a значение переменной b , а потом переменной b присвоить значение переменной c .

Блок-схема такого алгоритма показана на рис. 2.2.

Текст программы на «псевдоязыке»:

```
ввод
  a , b
  c=a
```

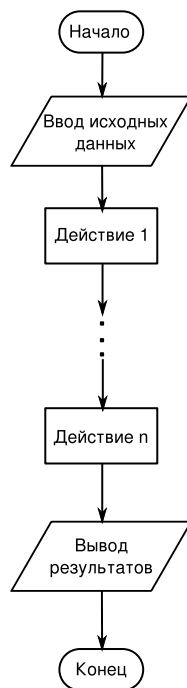



Рис. 2.1: Типичная схема линейного алгоритма

```

a=b
b=свывод
a , b

```

Метод решения с использованием особенностей Python: использовать два кортежа. В первом будут определены переменные `a` и `b` и их значения, а второй сформируем из этих же переменных, но в обратном порядке.

Текст программы на Python:

```

# -*- coding: utf-8 -*-
# Перестановка местами двух чисел с использованием кортежа
#
(a , b)=input ( 'Введите исходные значения (a , b) через запятую: ' )
(a , b) = (b , a)
print 'Новое значение a: ' , a , '\n' , 'Новое значение b: ' , b

```

Как описано в разделе 1.4.2, комбинация `'\n'` означает директиву на перевод строки для команды **print**.

Задача 2. Известны оклад (зарплата) и ставка процента подоходного налога. Определить размер подоходного налога и сумму, получаемую на руки.

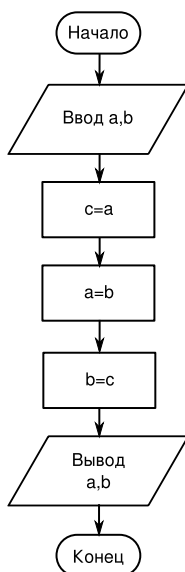


Рис. 2.2: Блок-схема алгоритма обмена значениями

Постановка задачи: Исходными данными являются величина оклада (переменная *oklad*, выражаемая числом) и ставка подоходного налога (переменная *procent*, выражаемая числом). Размер налога (переменная *nalog*) определяется как $oklad * procent / 100$, а сумма, получаемая на руки (переменная *summa*) — как $oklad - nalog$.

Блок-схема алгоритма показана на рис. 2.3.

Текст программы на «псевдоязыке»:

```
ввод oklad , procent
nalog=oklad*procent/100
summa=oklad-nalog
вывод summa , nalog
```

Программа на Python:

```
# -*- coding: utf-8 -*-
#
oklad=input("Оклад: ")
procent=input("% налога: ")
nalog=float(oklad*procent)/100
summa=oklad-nalog
print "Сумма на руки: ",summa
print "Налог: ",nalog
```

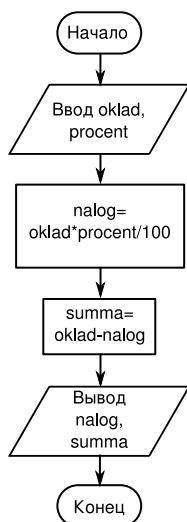


Рис. 2.3: Блок-схема задачи о налоге

Если все числа в этом примере использовать как целые, то результат может получиться неверным. Поэтому при вычислении налога используется преобразование числителя из целого числа в вещественное (функция `float()`).

Задача 3. Используя данные таблицы

Блюдо	Цена
Борщ	35
Котлета	40
Каша	20
Чай	3

определить общую стоимость обеда в столовой. Определить, во сколько раз возрастёт стоимость обеда, если цена котлеты увеличится вдвое¹

Постановка задачи (формализованная): Имеется четыре числа, которые требуется просуммировать (обозначим их переменными a , b , c и d соответственно). Сумму их значений обозначим $S1$. Требуется найти также величину $S2 = S1 + b$ и определить отношение $S2/S1$ (обозначим это отношение переменной res). В результате нужно вывести значения переменных $S1$ и res .

Блок-схема показана на рис. 2.4

Текст программы на «псевдоязыке»:

ввод a, b, c, d

¹Источник: В.А.Молодцов, Н.Б.Рыжикова. Информатика: тесты, задания, лучшие методики. Ростов-на-Дону: Феникс, 2009.

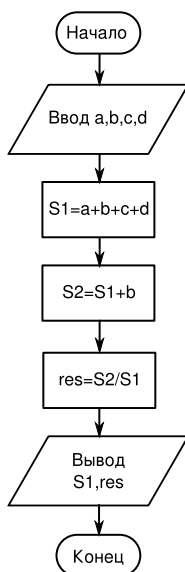


Рис. 2.4: Блок-схема задачи об обеде

```

S1=a , b , c , d
S2=S1+b
res=S2/S1
вывод S1 , res

```

В программе на Python разумно будет использовать кортеж:

```

# -*- coding: utf-8 -*-
#
t=(a , b , c , d)=input ( 'Введите значения через запятую: ' )
S1=sum( t )
S2=S1+b
res=float( S2 )/S1
print 'Начальная стоимость: ' , S1 , '\n' , 'Увеличение , раз: ' , res

```

И снова для преобразования целого числа в вещественное использована функция `float()`. (Полезно сравнить результат, получаемый при использовании выражения `res=float(S2)/S1` и выражения `res=float(S2/S1)`).

Задача 4. Преобразовать дату в «компьютерном» представлении (системную дату) в «русский» формат, т.е. день/месяц/год (например, 17/05/2009).

Постановка задачи: Системная дата имеет вид 2009-06-15. Нужно преобразовать это значение в строку, строку разделить на компоненты (символ-

разделитель — дефис), потом из этих компонентов сконструировать нужную строку.

Сразу перейдём к программе на Python. Функциями работы с датами и временем в Python «заведует» модуль `datetime`, а непосредственно для работы с датами используется объект `date` и его методы.

Воспользуемся знанием методов строк и списков.

```
# -*- coding: utf-8 -*-
#
# Подключаем нужный программный модуль
from datetime import date
# Получаем текущую дату
d1=date.today()
# Преобразуем результат в строку
ds=str(d1)
print "Системная_дата_", ds
# Используем методы строки и списка
lst=ds.split('-')
lst.reverse()
# Составляем новую строку для даты
rusdate="/".join(lst)
print "Российский_стандарт_", rusdate
```

Комментарии в тексте программы помогают понять происходящее.

2.1.1 Задачи для самостоятельного решения

1. Нарисуйте блок-схему к задаче 4 этой главы.
2. Даны действительные числа A,B,C. Найти максимальное и минимальное из этих чисел.
3. Известны длины трёх сторон треугольника. Вычислить периметр треугольника и площадь по формуле Герона (указание: использовать модуль `math` и функцию `sqrt()`).
4. Задан вес в граммах. Определить вес в тоннах и килограммах.
5. Известен объем информации в байтах. Перевести в килобайты, мегабайты.
6. Определить значение функции $Z=1/(XY)$ при X и Y не равных 0.

2.2 Ветвления и оператор выбора

В решениях задач по алгоритмизации одним из важнейших элементов является так называемое «ветвление», которое хорошо описывается сказочной фор-

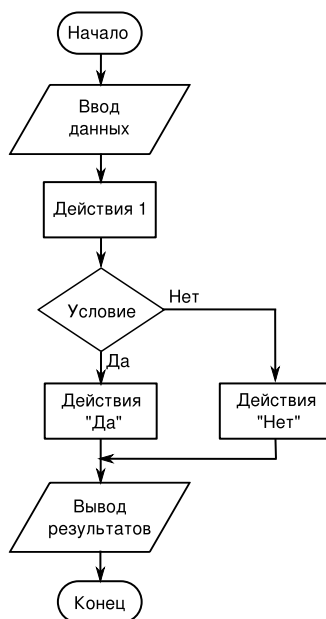


Рис. 2.5: Типовая схема алгоритма с ветвлением

мулой «Направо пойдёшь — голову потеряешь, прямо пойдёшь — коня потеряешь...», а проще говоря, ситуация «если ..., то ..., иначе ...». Типовая блок-схема алгоритма с ветвлением (проверкой условия) показана на рис. 2.5.

Если условие, указанное в блоке «Условие», выполняется, то далее производятся действия, соответствующие «ветви ДА» («Действия ДА»), иначе выполняются действия, соответствующие «ветви НЕТ» («Действия НЕТ»). Условия нужно составлять так, чтобы результат проверки любого условия допускал только два исхода — условие либо выполняется, либо не выполняется.

В случае, когда одной проверкой не удаётся охватить все варианты, используются «вложенные» условия, как показано на рис. 2.6. Условия могут быть вложены друг в друга любое количество раз (уровень вложенности не ограничен). Такая ситуация также называется «выбор».

В языках программирования для обеспечения проверки условий используется специальный составной оператор IF («если»). В этом операторе указывается условие, которое нужно проверить, и действия для ветвей «ДА» и «НЕТ».

Чтобы понять, как работает оператор IF, рассмотрим типичные задачи на проверку условий и выбор.

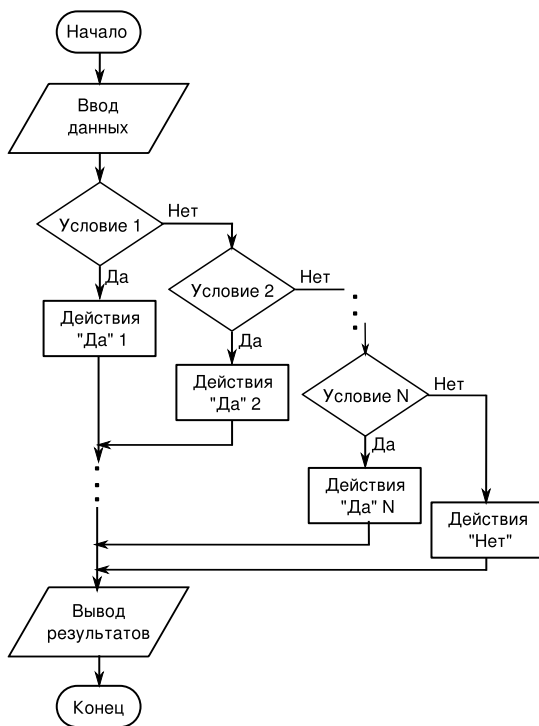


Рис. 2.6: Блок-схема алгоритма выбора

Задача 1. Составить программу ввода значения температуры воздуха t и выдачи текста «Хорошая погода!», если $t > 10$ градусов и текста «Плохая погода!», если $t \leq 10$ градусов².

Постановка задачи: Исходными данными является значение t , необходимо сформировать строку s . При $t < 10$ $s = \text{«Плохая_погода!»}$, иначе $s = \text{«Хорошая_погода!»}$.

Блок-схема алгоритма показана на рис. 2.7.

Текст программы на «псевдоязыке»:

```

ввод t
если (t < 10) то
    s = 'Плохая_погода!'
иначе
    s = 'Хорошая_погода!'
конец если
вывод s
  
```

²Источник: В.А.Молодцов, Н.Б.Рыжикова. Информатика: тесты, задания, лучшие методики. Ростов-на-Дону: Феникс, 2009.

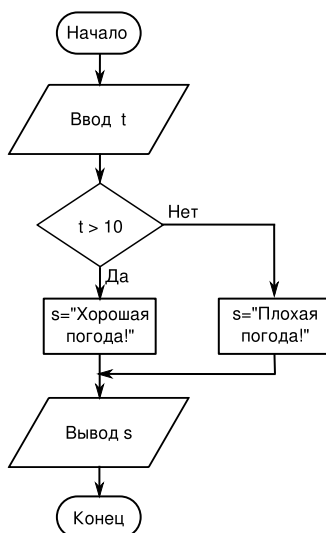


Рис. 2.7: Блок-схема алгоритма задачи про погоду

Текст на Python:

```

# -*- coding: utf-8 -*-
#
t=input( 'Введите_температуру_в_градусах:_ ' )
if t<10:
    s='Плохая_погода! '
else:
    s='Хорошая_погода! '
print s

```

Начало каждой «ветви» программы обозначается символом «:». Условие в операторе IF («если») записывается без скобок. Как таковое окончание оператора IF отсутствует. Python считает, что следующий оператор начинается в строке без отступа. Таким образом, в Python отступы играют важную роль.

Задача 2 (источник тот же). Составить программу ввода оценки P , полученной учащимся, и выдачи текста «Молодец!», если $P = 5$, «Хорошо!», если $P = 4$ и «Лентяй!», если $P \leq 3$.

Постановка задачи: Дано значение P , которое является натуральным числом и не может быть больше 5. В зависимости от величины P нужно сформировать строку s по правилам, указанным в условии. Необходимо выполнить две последовательные проверки значения P .

Блок-схема алгоритма показана на рис. 2.8.

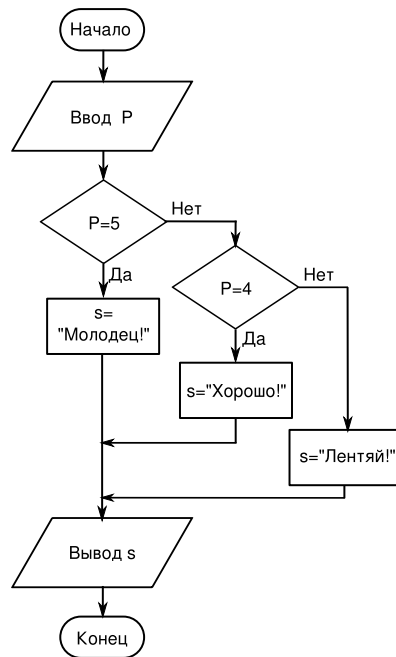


Рис. 2.8: Блок-схема алгоритма к задаче про оценки

Текст программы на «псевдоязыке»:

```

ввод P
если (P=5) то
    s='Молодец!'
иначе если (P=4)
    s='Хорошо!'
иначе
    s='Лентяй!'
конец если
вывод s
  
```

Программа на Python:

```

# -*- coding: utf-8 -*-
#
P=input('Ваши баллы? ')
if P==5:
    s='Молодец!'
elif P==4:
    s='Хорошо!'
  
```

```
else :  
    s= 'Лентяй! '  
print s
```

Ключевое слово **elif** в Python является сокращением от **else if** («иначе если») и используется для организации вложенных условий (алгоритмов выбора).

2.2.1 Задачи для самостоятельного решения

1. Дано натуральное число. Определить, будет ли это число: чётным, кратным 4.
2. Дано натуральное число. Определить, будет ли это число: нечётным, кратным 5.
3. Дано натуральное число. Определить, будет ли это число: нечётным, кратным 7.
4. Дано натуральное число. Определить, будет ли это число: чётным, кратным 10.
5. Имеется коробка со сторонами: $A \times B \times C$. Определить, пройдёт ли она в дверь с размерами $M \times K$.
6. Дано вещественное число. Определить, какое это число: положительное, отрицательное, ноль.
7. Можно ли из бревна, имеющего диаметр поперечного сечения D , выпилить квадратный брус шириной A ?
8. Можно ли в квадратном зале площадью S поместить круглую сцену радиусом R так, чтобы от стены до сцены был проход не менее K ?
9. Дан номер места в плацкартном вагоне. Определить, какое это место: верхнее или нижнее, в купе или боковое.
10. Известна денежная сумма. Разменять её купюрами 500, 100, 10 и монетой 2 руб., если это возможно.
11. Имеются две ёмкости: кубическая с ребром A , цилиндрическая с высотой H и радиусом основания R . Определить, поместится ли жидкость объёма M в первую ёмкость, во вторую, в обе.
12. Имеются две ёмкости: кубическая с ребром A , цилиндрическая с высотой H и радиусом основания R . Определить, можно ли заполнить жидкостью объёма M первую ёмкость, вторую, обе.

13. Даны вещественные числа: X, Y, Z . Определить, существует ли треугольник с такими длинами сторон и, если существует, будет ли он прямоугольным.
14. Дано число X . Определить, принадлежит ли это число заданному промежутку $[a, b]$.
15. Определить значение функции $Z = 1/(XY)$ при произвольных X и Y .
16. Даны вещественные числа: A, B, C . Определить, выполняются ли неравенства $A < B < C$ или $A \geq B \geq C$ и какое именно неравенство выполняется.
17. Даны две вещественные числа X и Y . Вычислить Z . $Z = \sqrt{X * Y}$ при $X > Y$, $Z = \ln(X + Y)$ в противном случае.
18. Даны вещественные положительные числа a, b, c, d . Выясните, может ли прямоугольник со сторонами a, b уместиться внутри прямоугольника со сторонами c, d так, чтобы каждая сторона внутреннего прямоугольника была параллельна или перпендикулярна стороне внешнего прямоугольника.
19. Дано вещественное число A . Вычислить $f(A)$, если $f(x) = x^2 + 4x + 5$, при $x \leq 2$; в противном случае $f(x) = 1/(x^2 + 4x + 5)$.
20. Дано вещественное число A . Вычислить $f(A)$, если $f(x) = 0$, при $x \leq 0$; $f(x) = x$ при $0 < x \leq 1$, в противном случае $f(x) = x^4$.
21. Дано вещественное число A . Вычислить $f(A)$, если $f(x) = 0$ при $x \leq 0$; $f(x) = x^2 - x$ при $0 < x \leq 1$, в противном случае $f(x) = x^2 - \sin(\pi x^2)$.
22. Составить алгоритм и программу для реализации логических операций «И» и «ИЛИ» для двух переменных.
23. Известен ГОД. Определить, будет ли этот год високосным, и к какому веку этот год относится.

Указание. При вычислении корней и логарифмов используйте функции `sqrt()` и `log()` модуля `math`. В этом же модуле определена константа `pi` (`math.pi`).

2.3 Циклические алгоритмы. Обработка последовательностей и одномерных массивов

Циклом называется фрагмент алгоритма или программы, который может повторяться несколько раз (в том числе и нуль раз). Каждая циклическая конструкция начинается заголовком цикла и заканчивается конечным оператором. Между ними располагаются операторы, называемые «телом цикла». Количество

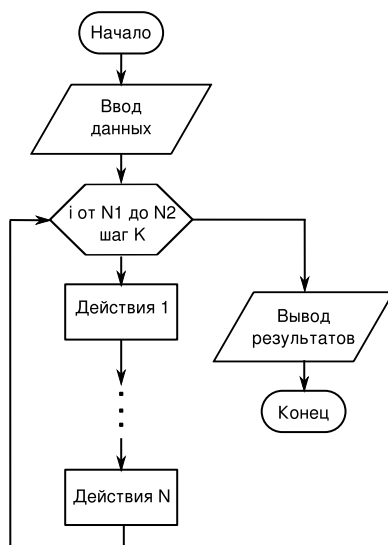


Рис. 2.9: Пример блок-схемы цикла с параметром

повторений выполнения команд (операторов), составляющих тело цикла, определяется условием окончания цикла. Условием окончания может быть достижение некоторого значения специальной переменной, называемой параметром цикла (переменной цикла), или выполнение (прекращение выполнения) некоторого условия.

Для организации циклов с параметром в языках программирования используется составной оператор FOR («для»), а в циклах с условием чаще всего используется составной оператор WHILE («пока»).

В случае цикла с параметром количество повторений («оборотов») цикла известно заранее и задаётся специальным выражением в заголовке цикла, а в случае цикла с условием при каждом следующем повторении требуется проверять условие прекращения цикла.

Если при написании операторов в теле цикла допущена ошибка, условие прекращения цикла может не выполниться никогда и цикл окажется бесконечным («программа зациклится»).

Пример блок-схемы цикла с параметром (переменной) показан на рис. 2.9, а пример блок-схемы цикла с условием окончания — на рис. 2.10. Для обозначения заголовка цикла с параметром используется специальный графический элемент — блок модификации, в котором указывается правило изменения параметра цикла.

Для работы с одномерными массивами целесообразно использовать циклы с параметром, поскольку до начала цикла может быть определено количество

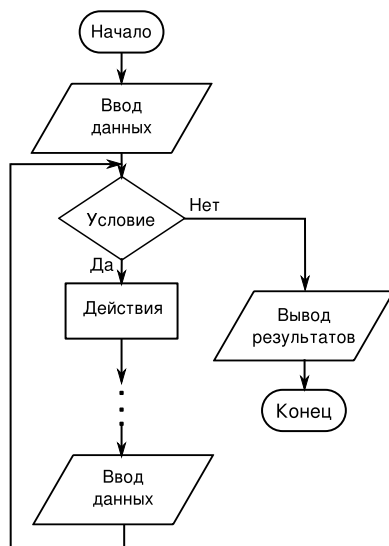


Рис. 2.10: Пример блок-схемы цикла с условием

повторений. В этом случае цикл с параметром требуется для ввода элементов массива, а для выполнения каких-либо действий с этими элементами и вывода результатов также могут потребоваться циклы.

В блок-схеме на рис. 2.10 действия повторяются, пока выполняется некоторое условие. Когда условие перестаёт выполняться, цикл завершается.

Такие циклы целесообразно использовать в ситуации, когда данные вводятся (поступают из какого-то источника), пока не произойдёт некоторое событие. При этом всю обработку чаще всего приходится выполнять «на лету», не создавая массив, поскольку количество элементов заранее неизвестно.

Рассмотрим типичные задачи, решение которых требует вычислений в цикле.

Задача 1. Дан одномерный массив A числовых значений, насчитывающий N элементов. Найти среднее арифметическое элементов массива.

Постановка задачи:

Дано:

N – количество элементов в массиве;

i – индекс элемента массива (параметр цикла).

$A[i]$ – элемент массива;

Найти:

S – сумма элементов массива

C – среднее арифметическое элементов массива, $C = S/N$.

Блок-схема алгоритма показана на рис. 2.11.

Текст программы на «псевдоязыке»:

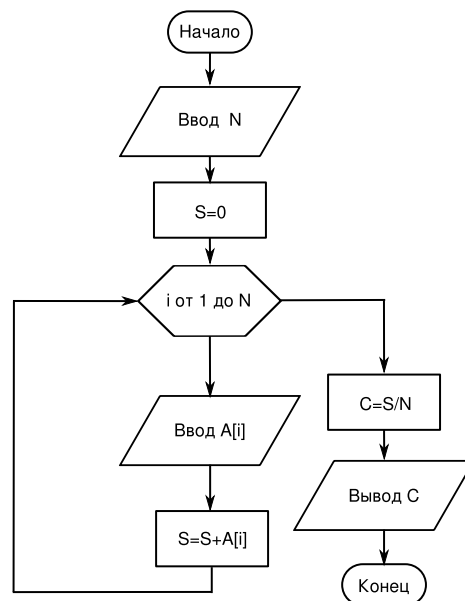


Рис. 2.11: Блок-схема алгоритма вычисления среднего значения в массиве

```

ввод N
S=0
нц для i от 1 до N
    ввод A[i]
    S=S+A[i]
кц
C=S/N
вывод C
  
```

Здесь **нц** и **кц** обозначают, соответственно, начало и конец цикла, строка с **нц** является заголовком цикла. Как видно из текста, указываются начальное и конечное значение переменной цикла, которая обязательно должна быть целым числом. В приведённой здесь записи переменная цикла увеличивается на 1 при каждом повторении («шаг переменной цикла» равен 1). Если требуется шаг, не равный 1, это указывается специально.

Тело цикла состоит из двух операторов — ввода очередного числа и прибавления этого числа к текущему значению суммы.

На Python можно написать практически то же самое (с учётом особенностей, связанных с использованием функции `range()`).

```

# -*- coding: utf-8 -*-
#
  
```

```

N=input( 'Количество_элементов:_ ' )
S=0
for i in range(N-1):
    a=input( 'Введите_число:_ ' )
    S=S+a
C=S/N
print 'Результат: ',C

```

Поскольку диапазон чисел, формируемых функцией `range()`, начинается с 0, то верхней границей должно быть $N-1$. Так как массив хранить нет необходимости, можно просто вводить числа и добавлять их к текущему значению суммы.

Тело цикла начинается после символа «:», и все операторы тела цикла в Python должны иметь одинаковый отступ от начала строки. Как только отступ исчезает, Python считает, что тело цикла закончилось.

А вот вариант решения этой же задачи на Python с использованием списка и методов списка.

```

# -*- coding: utf-8 -*-
#
N=input( 'Количество_элементов:_ ' )
S=0
lst=[]
for i in range(N-1):
    a=input( 'Введите_число:_ ' )
    lst.append(a)
C=sum( lst )/N
print 'Результат: ',C

```

В этом варианте формируется список, а сумма элементов списка вычисляется с помощью встроенной функции. Программа увеличилась на одну строку (создание пустого списка), но зато мы научились формировать список в цикле.

Задача 2. Определить, является ли введённая строка палиндромом («перевёртышем») типа АВВА, казак и пр.

Постановка задачи: Требуется сравнивать попарно символы с начала и с конца строки S (первый и последний, второй и предпоследний и т.д.). Если в каждой такой паре символы одинаковы, строка является палиндромом. Соответственно, каждая проверка пары символов должна получить некоторый признак (`flag` — «флаг»), который будет равен 1, если символы в паре совпадают и 0, если не совпадают. Окончательный результат обработки строки получится как произведение всех значений «флагов». Если хотя бы один раз «флаг» оказался равен нулю, строка палиндромом не является и произведение всех «флагов» окажется равным 0. Количество пар не превышает половины длины строки L (точно равно половине длины для строк с чётным количеством символов и результат целочисленного деления длины строки на 2 для строк с нечётным количеством символов,

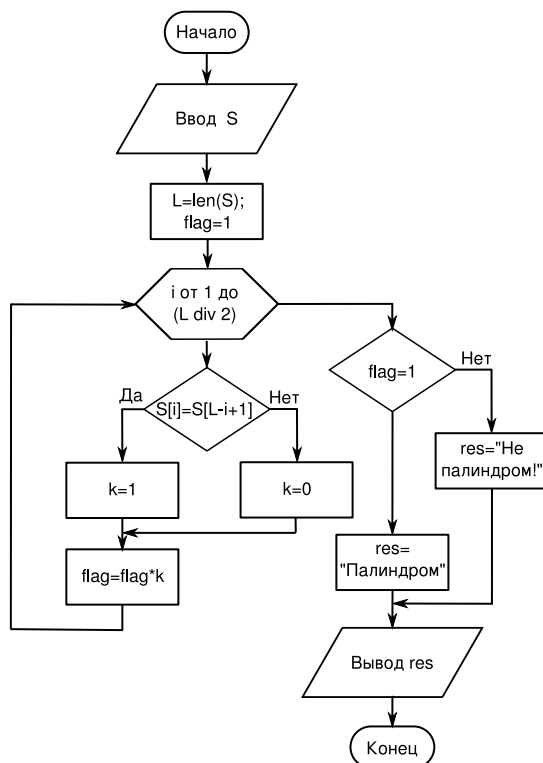


Рис. 2.12: Блок-схема алгоритма определения палиндрома

поскольку «центральный» символ строки с нечётным количеством символов очевидно совпадает сам с собой).

Блок-схема алгоритма показана на рис. 2.12.

Текст программы на «псевдоязыке»:

```

ввод S
flag=1
L=длина(S)
N=L div 2
нц для i от 1 до N
    если S[i]=S[L-i+1] то
        k=1
    иначе
        k=0
    конец если
    flag=flag*k
кц
  
```



```

если flag=1 to
    вывод 'Палиндром'
иначе
    вывод 'Не_палиндром!'
конец если

```

При проверке каждой пары устанавливается коэффициент k , который затем умножается на текущее значение «флага». Окончательный вывод делается по итоговому значению «флага».

Текст программы на Python может быть очень похож на текст на псевдоязыке.

```

# -*- coding: utf-8 -*-
#
s1=raw_input('Исходная_строка: ')
# Определяем длину строки
L=len(s1)
flag=1
for i in range(L//2):
    if s1[i]==s1[-i-1]:
        k=1
    else:
        k=0
    flag=flag*k
if flag==1:
    print 'Палиндром'
else:
    print 'Не_палиндром!'

```

Для ввода строки использован оператор `raw_input()`, при этом не требуется записывать строку в кавычках.

Небольшие синтаксические особенности всё-таки есть — условие равенства двух переменных записывается знаком «`==`», начало каждого составного оператора обозначается символом «`:`», и, как всегда, необходимо следить за отступами. Кроме того, чтобы отсчитывать символы с конца строки, использованы «отрицательные» индексы элементов строки.

Однако использование особенностей строк в Python, их функций и методов, позволяет решить эту задачу более изящно. Например, так.

```

# -*- coding: utf-8 -*-
#
s1=raw_input('Исходная_строка: ')
lst=list(s1)
lst.reverse()
s2=''.join(lst)
if s1==s2:

```

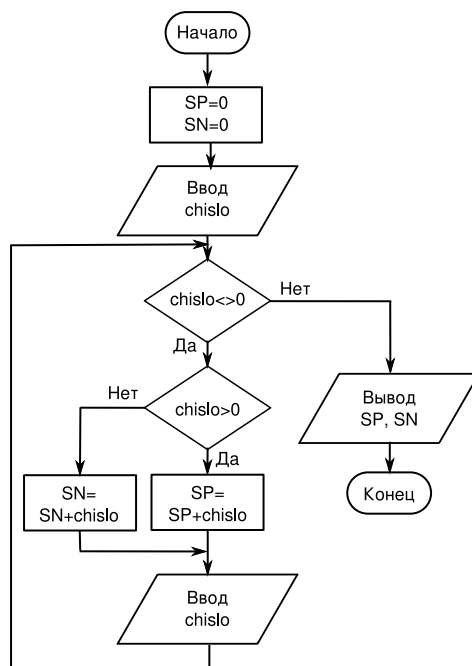


Рис. 2.13: Блок-схема алгоритма обработки последовательности

```

    print 'Палиндром'
else:
    print 'Не_палиндром!'

```

Здесь исходная строка преобразуется в список, затем список «переворачивается» и из него с помощью пустой «строки-объединителя» формируется новая строка. Затем строки сравниваются. Цикл оказывается не нужен! Всю работу делает Python.

Если количество повторений операций заранее неизвестно, но известно условие прекращения выполнения операций, используется цикл (составной оператор) WHILE. Покажем его использование на следующем примере.

Задача 3. Последовательно вводятся ненулевые числа. Определить сумму положительных и сумму отрицательных чисел. Закончить ввод чисел при вводе 0.

Задача настолько проста, что дополнительных уточнений в качестве постановки задачи не требуется. Пусть сумма положительных чисел называется SP, а сумма отрицательных чисел — SN.

Блок-схема алгоритма показана на рис. 2.13.

Текст программы на «псевдоязыке»:

```

SP=0
SN=0
ввод chislo
нц пока chislo <> 0
    если chislo > 0 то
        SP=SP+chislo
    иначе
        SN=SN+chislo
    конец если
вывод chislo
кц
вывод SP
вывод SN

```

Условие «неравенства» в языках программирования Pascal и BASIC обозначается как «<>», поэтому здесь сохранено это обозначение.

Следует обратить внимание, что проверяемое число нужно определить до начала цикла, поскольку возможна ситуация, что неопределённое значение окажется равным 0 и программа закончится, не успев начаться. А потом числа вводятся в цикле и каждое вновь поступившее число сравнивается с 0 (после ввода каждого числа следует проверка условия). Порядок операций и проверок в цикле WHILE может оказаться важным для получения верного результата.

Текст программы на Python не имеет каких-то существенных особенностей. Для удобства чтения программа поделена на «блоки» с помощью символа комментария.

```

# -*- coding: utf-8 -*-
#
SP=0
SN=0
#
chislo=input('Следующее_число: ')
#
while chislo != 0:
    if chislo > 0:
        SP=SP+chislo
    else:
        SN=SN+chislo
    chislo=input('Следующее_число: ')
#
print 'Сумма_положительных: ',SP
print 'Сумма_отрицательных: ',SN

```

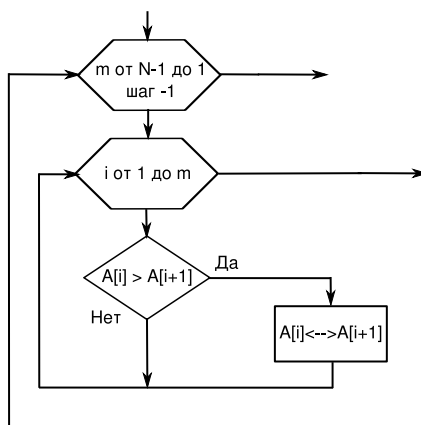


Рис. 2.14: Алгоритм сортировки «методом пузырька»

2.3.1 Сортировка массива

Задача сортировки, а также задача поиска максимального или минимального элемента в массиве встречается довольно часто. Средствами Python такие задачи решаются очень просто, но тем не менее рассмотрим общую задачу сортировки массива.

Под сортировкой понимается процедура, в результате выполнения которой изменяется исходный порядок следования данных. Причём новый порядок их следования отвечает требованию возрастания или убывания значений элементов одномерного массива. Например, при сортировке по возрастанию из одномерного массива $[3\ 1\ 0\ 5\ 2\ 7]$ получается массив $[0\ 1\ 2\ 3\ 5\ 7]$. Возможны и более сложные критерии сортировки. Символьные данные обычно сортируются в алфавитном порядке.

Один из наиболее наглядных методов сортировки — «метод пузырька».

Пусть необходимо упорядочить элементы массива A из N элементов по возрастанию.

Просматривая элементы массива «слева направо» (от первого элемента к последнему), меняем местами значения каждой пары соседних элементов в случае неравенства $A[i] > A[i + 1]$, передвигая тем самым наибольшее значение на последнее место. Следующие просмотры начинаем опять с первого элемента массива, последовательно уменьшая на единицу количество просматриваемых элементов. Процесс заканчивается после $N - 1$ просмотра.

Метод получил такое название, потому что каждое наибольшее значение как бы всплывает вверх.

Фрагмент блок-схемы алгоритма показан на рис. 2.14.

Действие $A[i] \leftrightarrow A[i + 1]$ означает перестановку значений элементов массива.

Текст соответствующего фрагмента программы на «псевдоязыке»:

```
ввод N, A
нц для m от N-1 до 1 шаг -1
    нц для i от 1 до m
        если A[i] > A[i+1] то
            X=A[i]
            A[i]=B[i+1]
            A[i+1]=X
        конец если
    кц
кц
вывод A
```

В этом фрагменте для перестановки значений элементов массива используется промежуточная переменная.

Задача поиска максимального элемента в массиве решается следующим образом. Пусть $\max A$ — требуемое значение максимального элемента. Сначала присваиваем переменной $\max A$ значение первого элемента массива, потом сравниваем первый элемент со следующим. Если следующий элемент (второй) больше первого, присваиваем его значение переменной $\max A$, а если нет — переходим к следующему (третьему элементу) и т. д.

Аналогично решается задача поиска минимального элемента в массиве.

В Python эти алгоритмы уже реализованы в функциях $\max()$, $\min()$ и в методе $\text{sort}()$ (метод $\text{sort}()$ сортирует список по возрастанию значений элементов).

2.3.2 Задачи для самостоятельного решения

1. Составьте блок-схему поиска максимального элемента в одномерном массиве.
2. Нарисуйте полную блок-схему алгоритма сортировки массива «методом пузырька».
3. Дан одномерный массив числовых значений, насчитывающий N элементов. Поменять местами элементы, стоящие на чётных и нечётных местах: $A[1] \leftrightarrow A[2]$; $A[3] \leftrightarrow A[4] \dots$
4. Дан одномерный массив числовых значений, насчитывающий N элементов. Выполнить перемещение элементов массива по кругу вправо, т. е. $A[1] \rightarrow A[2]$; $A[2] \rightarrow A[3]$; $\dots A[n] \rightarrow A[1]$.
5. Дан одномерный массив числовых значений, насчитывающий N элементов. Поменять местами первую и вторую половины массива.
6. Дан одномерный массив числовых значений, насчитывающий N элементов. Поменять местами группу из M элементов, начинающихся с позиции K с группой из M элементов, начинающихся с позиции P .

7. Дан одномерный массив числовых значений, насчитывающий N элементов. Вставить группу из M новых элементов, начиная с позиции K .
8. Дан одномерный массив числовых значений, насчитывающий N элементов. Сумму элементов массива и количество положительных элементов поставить на первое и второе место.
9. Дан одномерный массив числовых значений, насчитывающий N элементов. Исключить из него M элементов, начиная с позиции K .
10. Дан одномерный массив числовых значений, насчитывающий N элементов. Исключить все нулевые элементы.
11. Дан одномерный массив числовых значений, насчитывающий N элементов. После каждого отрицательного элемента вставить новый элемент, равный квадрату этого отрицательного элемента.
12. Дан одномерный массив числовых значений, насчитывающий N элементов. Определить, образуют ли элементы массива, расположенные перед первым отрицательным элементом, возрастающую последовательность.
13. Дан одномерный массив числовых значений, насчитывающий N элементов. Определить, образуют ли элементы массива, расположенные перед первым отрицательным элементом, убывающую последовательность.
14. Дан одномерный массив числовых значений, насчитывающий N элементов. Из элементов исходного массива построить два новых. В первый должны входить только элементы с положительными значениями, а во второй — только элементы с отрицательными значениями.
15. Дан одномерный массив числовых значений, насчитывающий N элементов. Добавить столько элементов, чтобы элементов с положительными и отрицательными значениями стало бы поровну.
16. Дан одномерный массив числовых значений, насчитывающий N элементов. Добавить к элементам массива такой новый элемент, чтобы сумма элементов с положительными значениями стала бы равна модулю суммы элементов с отрицательными значениями.
17. Дан одномерный массив числовых значений, насчитывающий N элементов. Дано положительное число T . Разделить это число между положительными элементами массива пропорционально значениям этих элементов и добавить полученные доли к соответствующим элементам.
18. Дан одномерный массив числовых значений, насчитывающий N элементов. Исключить из массива элементы, принадлежащие промежутку $[B; C]$.

19. Дан одномерный массив числовых значений, насчитывающий N элементов. Вместо каждого элемента с нулевым значением поставить сумму двух предыдущих элементов массива.
20. Дан одномерный массив числовых значений, насчитывающий N элементов. Определить, имеются ли в массиве два подряд идущих нуля.
21. Дан одномерный массив числовых значений, насчитывающий N элементов. Подсчитать количество чисел, делящихся на 3 нацело, и среднее арифметическое чисел с чётными значениями. Поставить полученные величины на первое и последнее места в массиве (увеличив массив на 2 элемента).
22. Заданы M строк символов, которые вводятся с клавиатуры. Найти количество символов в самой длинной строке. Выровнять строки по самой длинной строке, поставив перед каждой строкой соответствующее количество звёздочек.
23. Заданы M строк символов, которые вводятся с клавиатуры. Из заданных строк, каждая из которых представляет одно слово, составить одну длинную строку, разделяя слова пробелами.
24. Заданы M строк слов, которые вводятся с клавиатуры. Подсчитать количество гласных букв в каждой из заданных строк.
25. Заданы M строк слов, которые вводятся с клавиатуры (в каждой строке – одно слово). Вводится слог (последовательность букв). Подсчитать количество таких слогов в каждой строке.
26. Заданы M строк слов, которые вводятся с клавиатуры (в каждой строке – одно слово). Вводится слог (последовательность букв). Удалить данный слог из каждой строки.
27. Заданы M строк символов, которые вводятся с клавиатуры. Напечатать все центральные буквы строк нечетной длины.
28. Заданы M строк символов, которые вводятся с клавиатуры. Каждая строка содержит слово. Записать каждое слово в разрядку (вставить по пробелу между буквами).
29. Задана строка символов, в которой встречается символ «.». Поставить после каждого такого символа системное время ПК.
30. Заданы M строк, которые вводятся с клавиатуры. Подсчитать количество пробелов в каждой из строк.
31. Заданы M строк символов, которые вводятся с клавиатуры. Каждая строка представляет собой последовательность символов, включающих в себя вопросительные знаки. Заменить в каждой строке все имеющиеся вопросительные знаки звёздочками.

32. Последовательно вводятся числа. Определить сумму чисел с нечётными номерами и произведение чисел с чётными номерами (по порядку ввода). Подсчитать количество слагаемых и количество сомножителей. При вводе числа 55555 закончить работу.
33. Определить сумму вводимых положительных чисел. Причём числа с нечётными номерами (по порядку ввода) суммировать с обратным знаком, а числа с чётными номерами перед суммированием возводить в квадрат. Подсчитать количество слагаемых. При вводе первого отрицательного числа закончить работу.
34. Даны число P и число H . Определить сумму чисел меньше P , произведение чисел больше H и количество чисел в диапазоне значений P и H . При вводе числа равного P или H , закончить работу.
35. Суммировать вводимые числа, среди которых нет нулевых. При вводе нуля обеспечить вывод текущего значения суммы. При вводе числа 99999 закончить работу.
36. Вводятся положительные числа. Определить сумму чисел, делящихся на положительное число B нацело. При вводе отрицательного числа закончить работу.
37. Для вводимых чисел определить процент положительных и отрицательных чисел. При вводе числа -65432 закончить работу.

2.4 Обработка двумерных массивов (матриц)

Двумерные массивы являются аналогами матриц и имеют «прямоугольную» (табличную) структуру. Описываются массивы так же, как одномерные. Разница состоит в том, что у элемента двумерного массива *две* координаты (два индекса) — номер строки и номер столбца, в которых находится элемент.

Ввод массива осуществляется построчно при помощи двух циклов. Пусть M — количество столбцов, N — количество строк. Элементы массива обозначим как $mas[i, j]$, первый индекс — номер строки, второй — номер столбца.

```
ввод M, N
нц для i от 1 до N
    нц для j от 1 до M
        ввод mas[i, j]
    кц
кц
```

Вывод массива на экран осуществляется при помощи аналогичных циклов.

```
нц для i от 1 до N
    нц для j от 1 до M
```



```

        вывод mas[i , j]
    кц
    вывод
кц

```

Здесь «пустой» оператор вывода обеспечивает переход на новую строку.

В Python для работы с многомерными (когда используется два и более индексов) массивами можно использовать вложенные списки (списки списков, списки списков списков и т. д.).

Однако Python предоставляет более удобный инструмент создания и преобразования многомерных массивов — библиотеку `numpy` (Numeric Python).

Создание двумерного массива в Python может выглядеть так:

```

# -*- coding: utf-8 -*-
#
import numpy
n=input ( 'Количество_строк:_ ' )
m=input ( 'Количество_столбцов:_ ' )
# Создаём "нулевую" матрицу
a=numpy.zeros ( [n-1,m-1])
# Заполняем матрицу
for i in range(n-1):
    for j in range(m-1):
        print 'Элемент_матрицы_[ ' , i , ' ] [ ' , j , ' ] '
        a [ i , j ]=input ( 'Введите_элемент:_ ' )
#

```

Сначала с помощью функции (метода) `numpy.zeros()` создаётся двумерный массив (матрица), заполненный нулями, а потом вместо нулей подставляются реальные значения. Индексы элементов, так же как в строках, кортежах и списках, начинаются с 0 (первый — верхний левый — элемент матрицы в Python имеет индекс [0,0]). Оператор **print** выводит индексы очередного элемента матрицы, который нужно ввести.

Задача 1. Выполнить обработку элементов прямоугольной матрицы A , имеющей N строк и M столбцов. Найти среднее арифметическое элементов массива.

Постановка задачи:

Дано:

n — количество строк в массиве;

m — количество столбцов в массиве;

$A[i, j]$ — элемент массива;

i, j — индексы элемента массива.

Найти:

S — сумма элементов массива (сумма всех $A[i, j]$ при всех i и j)

K — количество элементов в массиве ($K = m * n$)

C — среднее арифметическое элементов массива ($C = S/K$)

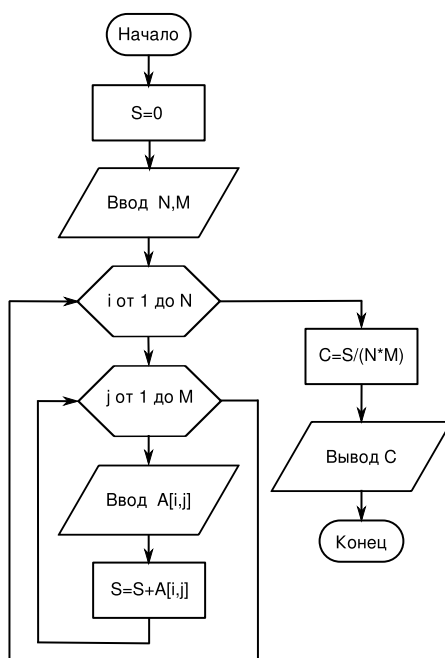


Рис. 2.15: Блок-схема алгоритма вычисления среднего значения матрицы

Блок-схема алгоритма решения показана на рис. 2.15.

Текст программы на «псевдоязыке»:

```

ввод n , m
S=0
нц для i от 1 до n
  нц для j от 1 до m
    ввод A[i , j]
    S=S+A[i , j]
  кц
кц
K=n*m
C=S/K
вывод C

```

Текст программы на Python:

```
# -*- coding: utf-8 -*-
#
import numpy
n=input('Количество_строк:_')
m=input('Количество_столбцов:_')
S=0.0
# Создаём нулевую матрицу
a=numpy.zeros([n-1,m-1])
# Заполняем матрицу
for i in range(n-1):
    for j in range(m-1):
        print 'Элемент_матрицы_[' , i , ' ][ ' , j , ' ] '
        a[i , j]=input('Введите_элемент:_')
        S=S+a[i , j]
#
K=n*m
C=S/K
print 'Среднее_значение_по_строкам: ', C
```

2.4.1 Задачи для самостоятельного решения

1. Выполнить обработку элементов прямоугольной матрицы A , имеющей N строк и M столбцов. Найти наибольший элемент столбца матрицы A , для которого сумма абсолютных значений элементов максимальна.
2. Выполнить обработку элементов прямоугольной матрицы A , имеющей N строк и M столбцов. Найти наибольшее значение среди средних значений для каждой строки матрицы.
3. Выполнить обработку элементов прямоугольной матрицы A , имеющей N строк и M столбцов. Найти наименьший элемент столбца матрицы A , для которого сумма абсолютных значений элементов максимальна.
4. Выполнить обработку элементов прямоугольной матрицы A , имеющей N строк и M столбцов. Найти наименьшее значение среди средних значений для каждой строки матрицы.
5. Выполнить обработку элементов прямоугольной матрицы A , имеющей N строк и M столбцов. Определить средние значения по всем строкам и столбцам матрицы. Результат оформить в виде матрицы из $N + 1$ строк и $M + 1$ столбцов.
6. Выполнить обработку элементов прямоугольной матрицы A , имеющей N строк и M столбцов. Найти сумму элементов всей матрицы. Определить,

какую долю в этой сумме составляет сумма элементов каждого столбца. Результат оформить в виде матрицы из $N + 1$ строк и M столбцов.

7. Выполнить обработку элементов прямоугольной матрицы A , имеющей N строк и M столбцов. Найти сумму элементов всей матрицы. Определить, какую долю в этой сумме составляет сумма элементов каждой строки. Результат оформить в виде матрицы из N строк и $M+1$ столбцов.
8. Выполнить обработку элементов прямоугольной матрицы A , имеющей N строк и M столбцов. Определить, сколько отрицательных элементов содержится в каждом столбце и в каждой строке матрицы. Результат оформить в виде матрицы из $N + 1$ строк и $M + 1$ столбцов.
9. Выполнить обработку элементов прямоугольной матрицы A , имеющей N строк и M столбцов. Определить, сколько нулевых элементов содержится в верхних L строках матрицы и в левых K столбцах матрицы.
10. Выполнить обработку элементов прямоугольной матрицы A , имеющей N строк и M столбцов. Перемножить элементы каждого столбца матрицы с соответствующими элементами K -го столбца.
11. Выполнить обработку элементов прямоугольной матрицы A , имеющей N строк и M столбцов. Просуммировать элементы каждой строки матрицы с соответствующими элементами L -й строки.
12. Выполнить обработку элементов прямоугольной матрицы A , имеющей N строк и M столбцов. Разделить элементы каждой строки на элемент этой строки с наибольшим значением.
13. Выполнить обработку элементов прямоугольной матрицы A , имеющей N строк и M столбцов. Разделить элементы каждого столбца матрицы на элемент этого столбца с наибольшим значением.
14. Выполнить обработку элементов прямоугольной матрицы A , имеющей N строк и M столбцов. Разделить элементы матрицы на элемент матрицы с наибольшим значением.
15. Выполнить обработку элементов прямоугольной матрицы A , имеющей N строк и M столбцов. Все элементы имеют целый тип. Дано целое число H . Определить, какие столбцы имеют хотя бы одно такое число, а какие не имеют.
16. Выполнить обработку элементов прямоугольной матрицы A , имеющей N строк и M столбцов. Исключить из матрицы строку с номером L . Сомкнуть строки матрицы.

17. Выполнить обработку элементов прямоугольной матрицы A , имеющей N строк и M столбцов. Добавить к матрице строку и вставить ее под номером L .
18. Выполнить обработку элементов квадратной матрицы A , имеющей N строк и N столбцов. Найти сумму элементов, стоящих на главной диагонали, и сумму элементов, стоящих на побочной диагонали (элементы главной диагонали имеют индексы от $[0,0]$ до $[N,N]$, а элементы побочной диагонали — от $[N,0]$ до $[0,N]$).
19. Выполнить обработку элементов квадратной матрицы A , имеющей N строк и N столбцов. Определить сумму элементов, расположенных параллельно главной диагонали (ближайшие к главной). Элементы главной диагонали имеют индексы от $[0,0]$ до $[N,N]$.
20. Выполнить обработку элементов квадратной матрицы A , имеющей N строк и N столбцов. Определить произведение элементов, расположенных параллельно побочной диагонали (ближайшие к побочной). Элементы побочной диагонали имеют индексы от $[N,0]$ до $[0,N]$.
21. Выполнить обработку элементов квадратной матрицы A , имеющей N строк и N столбцов. Каждой паре элементов, симметричных относительно главной диагонали (ближайшие к главной), присвоить значения, равные полусумме этих симметричных значений (элементы главной диагонали имеют индексы от $[0,0]$ до $[N,N]$).
22. Выполнить обработку элементов прямоугольной матрицы A , имеющей N строк и M столбцов. Исходная матрица состоит из нулей и единиц. Добавить к матрице еще один столбец, каждый элемент которого делает количество единиц в каждой строке чётным.
23. Выполнить обработку элементов квадратной матрицы A , имеющей N строк и N столбцов. Найти сумму элементов, расположенных выше главной диагонали, и произведение элементов, расположенных выше побочной диагонали (элементы главной диагонали имеют индексы от $[0,0]$ до $[N,N]$, а элементы побочной диагонали — от $[N,0]$ до $[0,N]$).
24. Выполнить обработку элементов прямоугольной матрицы A , имеющей N строк и M столбцов. Дан номер строки L и номер столбца K , при помощи которых исходная матрица разбивается на четыре части. Найти сумму элементов каждой части.
25. Выполнить обработку элементов прямоугольной матрицы A , имеющей N строк и M столбцов. Определить, сколько нулевых элементов содержится в каждом столбце и в каждой строке матрицы. Результат оформить в виде матрицы из $N + 1$ строк и $M + 1$ столбцов.

26. Выполнить обработку элементов прямоугольной матрицы A , имеющей N строк и M столбцов. Дан номер строки L и номер столбца K , при помощи которых исходная матрица разбивается на четыре части. Найти среднее арифметическое элементов каждой части.
27. Выполнить обработку элементов прямоугольной матрицы A , имеющей N строк и M столбцов. Все элементы имеют целый тип. Дано целое число H . Определить, какие строки имеют хотя бы одно такое число, а какие не имеют.
28. Выполнить обработку элементов прямоугольной матрицы A , имеющей N строк и M столбцов. Исключить из матрицы столбец с номером K . Сократить столбцы матрицы.
29. Выполнить обработку элементов прямоугольной матрицы A , имеющей N строк и M столбцов. Добавить к матрице столбец чисел и вставить его под номером K .
30. Выполнить обработку элементов прямоугольной матрицы A , имеющей N строк и M столбцов. Добавить к элементам каждого столбца такой новый элемент, чтобы сумма положительных элементов стала бы равна модулю суммы отрицательных элементов. Результат оформить в виде матрицы из $N + 1$ строк и M столбцов.
31. Выполнить обработку элементов прямоугольной матрицы A , имеющей N строк и M столбцов. Добавить к элементам каждой строки такой новый элемент, чтобы сумма положительных элементов стала бы равна модулю суммы отрицательных элементов. Результат оформить в виде матрицы из N строк и $M + 1$ столбцов.

2.5 Работа с ассоциативными массивами (таблицами данных)

Ассоциативный массив лучше всего описывается табличным представлением данных, когда каждая строка таблицы описывает характеристики какого-то объекта из множества однородных объектов (типичный пример — список учеников, их домашних телефонов и адресов). Таким образом, по значению из первого столбца такой таблицы (ключу) можно однозначно определить значения из остальных столбцов, т. е. значение ключа ассоциируется с остальными характеристиками объекта (в случае ученика — по фамилии можно найти другую информацию).

Если в ассоциативном массиве только два столбца («ключ» и «значение»), то такой массив называется «хэш». Такие ассоциативные массивы очень часто используются в современных информационных системах (например, пары «логин—пароль»).

В области моделирования процессов и явлений часто встречаются задачи, в которых значению «ключа» соответствует несколько параметров (например, номеру химического элемента однозначно соответствует название, атомный вес, валентность, количество протонов и пр.). В таких задачах простые хэш-массивы использовать уже неудобно.

Эффективный алгоритм обработки ассоциативных массивов (поиска значений, добавления и удаления значений и ключей, сортировки и пр.) в значительной степени зависит от используемого языка программирования и определённых в этом языке типов и структур данных. Так, в языке программирования Basic ассоциативный массив образуется из нескольких согласованных одномерных массивов. В языке программирования Pascal для представления ассоциативных массивов используется структура данных «запись» (record). В Python для ассоциативных массивов определена специальная структура данных — словарь, но мы рассмотрим работу с ассоциативными массивами с помощью списков и функций работы со списками.

Рассмотрим задачу из области экономического анализа.

Оценить экономическую деятельность нескольких предприятий. Известны названия предприятий, значения планового объёма розничного товарооборота и значения фактического объёма розничного товарооборота.

Требуется определить:

1. процент выполнения плана каждым предприятием;
2. количество предприятий, недовыполнивших план;
3. наибольший плановый товарооборот;
4. упорядочить предприятия по возрастанию планового товарооборота.

Обозначим количество предприятий как k и сформируем три списка — список названий предприятий (пусть он называется `name`), список значений планового товарооборота (назовём его `plan`), список значений фактического товарооборота (с именем `fact`). На основании этих данных создадим список значений процентов выполнения плана (пусть он называется `procent`).

Количество предприятий, недовыполнивших план, будем определять в результате сравнения процента выполнения со 100 процентами в цикле по всем предприятиям.

Текст программы на Python может выглядеть, как показано ниже.

```
# -*- coding: utf-8 -*-  
#  
# k - количество предприятий  
# name - список названий предприятий  
# plan - список значений планового товарооборота  
# fact - список значений фактического товарооборота  
# procent - список значений % выполнения плана
```

```

#
k=input ("Количество_предприятий: ")
name=[]
plan=[]
fact=[]
#
for i in range(k):
    n=raw_input ("Название: ")
    name.append(n)
    p1=input ("План: ")
    plan.append(p1)
    p2=input ("Факт: ")
    fact.append(p2)
#
procent=map(lambda x,y: x*100/y, fact, plan)
fakty=zip(name, procent)
plany=zip(plan, name)
plany.sort()
print 16*"="
print "Процент_выполнения_плана_каждым_предприятием: "
#
nedo=0
for i in range(k):
    s1=fakty[i][0]
    s2=fakty[i][1]
    if s2 < 100:
        nedo=nedo+1
#
print s1, ": ", s2
print "Количество_предприятий, _недовыполнивших_план: ", nedo
print "Наибольший_плановый_товарооборот: ", max(plan)
#
print "Предприятия_по_возрастанию_плана: "
for i in range(k):
    s1=plany[i][1]
    s2=plany[i][0]
    print s1, ": ", s2

```

Здесь с помощью функции `map()` и «одноразовой» `lambda`-функции создаётся список процентов выполнения плана и с помощью функции `zip()` формируется два итоговых ассоциативных массива. Сортировка таких ассоциативных массивов производится «по первому столбику», поэтому важен порядок аргументов в функции `zip()`, а также порядок индексов при выводе результатов.

Пример решения задачи показан на рис. 2.16.


```

Количество предприятий: 4
Название: Фанта
План: 56
Факт: 58
Название: Прима
План: 49
Факт: 47
Название: Люкос
План: 112
Факт: 131
Название: Мона
План: 94
Факт: 88
=====
Процент выполнения плана каждым предприятием:
Фанта : 103
Прима : 95
Люкос : 116
Мона : 93
Количество предприятий, невыполнивших план: 2
Наибольший плановый товарооборот: 112
Предприятия по возрастанию плана:
Прима : 49
Фанта : 56
Мона : 94
Люкос : 112

-----
(program exited with code: 0)
Press return to continue

```

Рис. 2.16: Пример решения задачи с ассоциативным массивом

2.5.1 Задачи для самостоятельного решения

1. Используя данные таблицы	Блюдо	Цена	отсортировать блюда по возрастанию цены. Вывести отсортированный вариант списка блюд.
	Борщ	35	
	Котлета	40	
	Каша	20	
	Чай	3	

- Имеется список учеников и результаты трёх тестов (баллы от 0 до 100). Определить средний балл каждого ученика по трём тестам, вывести список учеников по убыванию среднего балла.
- Известны данные о количестве мальчиков и девочек в нескольких классах. Отсортировать названия классов по возрастанию процента мальчиков, определить количество классов, в которых мальчиков больше, чем девочек, и вывести названия этих классов отдельно.

4. Решить задачу, связанную с оценкой экономической деятельности группы предприятий на основе известных данных:

- название предприятий;
- плановый объем розничного товарооборота;
- фактический объем розничного товарооборота.

Требуется определить:

- a) процент выполнения плана каждым предприятием;
- b) количество предприятий, недовыполнивших план на 10% и более;
- c) наименьший плановый товарооборот;
- d) упорядочить предприятия по убыванию планового товарооборота.

Глава 3

Графика в Python и задачи моделирования

Python может работать с несколькими графическими библиотеками, обеспечивая создание сложных приложений с развитым графическим пользовательским интерфейсом. В этой части мы научимся пользоваться самыми простыми графическими возможностями Python — управлять исполнителем «черепашка» для создания графических примитивов и перемещаться на плоскости и использовать методы модуля Tkinter для задач моделирования математических функций и физических явлений.

3.1 Управление исполнителем «черепашка»

Исполнитель «черепашка» управляется командами относительных («вперёд-назад» и «направо-налево») и абсолютных («перейти в точку с координатами...») перемещений. Исполнитель представляет собой «перо», оставляющее след на плоскости рисования. Перо можно поднять, тогда при перемещении след оставаться не будет. Кроме того, для пера можно установить толщину и цвет. Все эти функции исполнителя обеспечиваются модулем `turtle` («черепаха»).

Приведённый ниже код создаёт графическое окно (рис. 3.1) и помещает перо («черепашку») в исходное положение.

```
# -*- coding: utf-8 -*-
import turtle
# Инициализация
turtle.reset()
# Здесь могут быть вычисления и команды рисования
# turtle._root.mainloop() # для Python 2.4.x и 2.5.x
# Эта команда показывает окно, пока его не закроют
turtle.mainloop() # для Python 2.6.x
```

Как видно из этого примера, способ вызова метода `mainloop()` для показа графического окна зависит от версии Python. Поэтому если не работает один



Рис. 3.1: Окно рисования модуля turtle

вариант, смело используйте другой. В остальных примерах будет использоваться вариант для Python 2.6.

Полученное окно имеет фиксированный размер, который зависит от версии Python, перо позиционируется в центре. Идея рисования заключается в перемещении пера («черепашки») в точки окна рисования с указанными координатами или в указанных направлениях на заданные расстояния, а также в проведении отрезков прямых, дуг и окружностей.

Текущее направление перемещение пера (соответствующее направлению «вперёд») указывается остриём стрелки изображения «черепашки».

Полный список команд управления «черепашкой» (и, соответственно, рисования), а также функций, обеспечиваемых модулем, можно получить, набрав в окне выполнения любой системы программирования на Python команду `help(' turtle ')`.

Список этот довольно длинный, а среди предоставляемых функций имеются также математические, поскольку они могут быть востребованы при вычислении параметров отрезков, дуг и окружностей.

Команды, обеспечивающие рисование, приведены ниже.

Команда	Назначение	Пример
<code>up()</code>	Поднятие «пера», чтобы не оставалось следа его при перемещении	<code>turtle .up()</code>
<code>down()</code>	Опускание «пера», чтобы при перемещении оставался след (рисовались линии)	<code>turtle .down()</code>
<code>goto(x,y)</code>	Перемещение «пера» в точку с координатами x, y в системе координат окна рисования	<code>turtle .goto(50,20)</code>

Команда	Назначение	Пример
<code>color('цвет')</code>	Установка цвета «пера» в значение, определяемое строкой цвета	<code>turtle.color('blue')</code> <code>turtle.color('#0000ff')</code>
<code>width(n)</code>	Установка толщины «пера» в точках экрана	<code>turtle.width(3)</code>
<code>forward(n)</code>	Передвижение «вперёд» (в направлении острия стрелки, см. рис. 3.1) на n точек	<code>turtle.forward(100)</code>
<code>backward(n)</code>	Передвижение «назад» на n точек	<code>turtle.backward(100)</code>
<code>right(k)</code>	Поворот направо (по часовой стрелке) на k единиц	<code>turtle.right(75)</code>
<code>left(k)</code>	Поворот налево (против часовой стрелки) на k единиц	<code>turtle.left(45)</code>
<code>radians()</code>	Установка единиц измерения углов в радианы	<code>turtle.radians()</code>
<code>degrees()</code>	Установка единиц измерения углов в градусы (включён по умолчанию)	<code>turtle.degrees()</code>
<code>circle(r)</code>	Рисование окружности радиусом $ r $ точек из текущей позиции «пера». Если r положительно, окружность рисуется против часовой стрелки, если отрицательно — по часовой стрелке.	<code>turtle.circle(40)</code> <code>turtle.circle(-50)</code>
<code>circle(r,k)</code>	Рисование дуги радиусом $ r $ точек и углом k единиц. Вариант команды <code>circle()</code>	<code>turtle.circle(40,45)</code> <code>turtle.circle(-50,275)</code>
<code>fill(flag)</code>	В зависимости от значения <code>flag</code> включается (<code>flag=1</code>) и выключается (<code>flag=0</code>) режим закрашивания областей. По умолчанию выключен.	Круг: <code>turtle.fill(1)</code> <code>turtle.circle(-50)</code> <code>turtle.fill(0)</code>
<code>write('строка')</code>	Вывод текста в текущей позиции пера	<code>turtle.write('Начало_координат!')</code>

Команда	Назначение	Пример
<code>tracer (flag)</code>	Включение (<code>flag=1</code>) и выключение (<code>flag=0</code>) режима отображения указателя «пера» («черепашки»). По умолчанию включён.	<code>turtle . tracer (0)</code>
<code>clear ()</code>	Очистка области рисования	<code>turtle . clear (0)</code>

При выключенном режиме отображения указателя «черепашки» рисование происходит значительно быстрее, чем при включённом.

Нужно заметить, что хотя углы поворота исполнителя изначально интерпретируются в градусах, при использовании тригонометрических функций модуля `turtle` (например, `turtle . sin ()`) аргументы этих функций воспринимаются как радианы.

Проделаем упражнение с целью определить систему координат окна рисования. Приведённый ниже код формирует картинку, показанную на рис. 3.2.

```
# -*- coding: utf-8 -*-
import turtle
#
turtle . reset ()
turtle . tracer (0)
turtle . color ( '#0000 ff ' )
#
turtle . write ( ' 0,0 ' )
#
turtle . up ()
x=-170
y=-120
coords=str(x)+", "+str(y)
turtle . goto (x,y)
turtle . write (coords)
#
x=130
y=100
coords=str(x)+", "+str(y)
turtle . goto (x,y)
turtle . write (coords)
#
x=0
y=-100
coords=str(x)+", "+str(y)
```

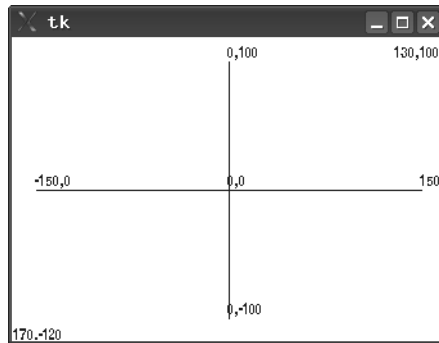


Рис. 3.2: Система координат окна рисования

```

turtle.goto(x,y)
turtle.write(coords)
#
turtle.down()
x=0
y=100
coords=str(x)+", "+str(y)
turtle.goto(x,y)
turtle.write(coords)
#
turtle.up()
x=-150
y=0
coords=str(x)+", "+str(y)
turtle.goto(x,y)
turtle.write(coords)
#
turtle.down()
x=150
y=0
coords=str(x)+", "+str(y)
turtle.goto(x,y)
turtle.write(coords)
#
turtle.mainloop()

```

Здесь строка с координатами формируется «в лоб», путём конкатенации преобразованных в строки значений координат.

Картинка, показанная на рис. 3.3, сформирована нижеследующим кодом.

```
# -*- coding: utf-8 -*-
import turtle
#
turtle.reset()
turtle.tracer(0)
turtle.width(2)
#
turtle.up()
x=0
y=-100
turtle.goto(x,y)
turtle.fill(1)
turtle.color('#ffaa00')
turtle.down()
turtle.circle(100)
turtle.fill(0)
turtle.color('black')
turtle.circle(100)
turtle.up()
#
x=-45
y=50
turtle.goto(x,y)
turtle.down()
turtle.color('#0000aa')
turtle.fill(1)
turtle.circle(7)
turtle.up()
turtle.fill(0)
#
x=45
y=50
turtle.goto(x,y)
turtle.down()
turtle.color('#0000aa')
turtle.fill(1)
turtle.circle(7)
turtle.up()
turtle.fill(0)
#
x=-55
y=-50
turtle.goto(x,y)
turtle.right(45)
```

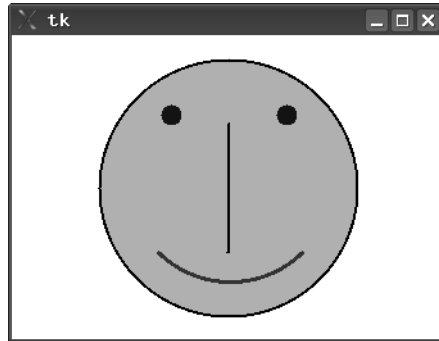



Рис. 3.3: Пример формирования изображения

```

turtle.width(3)
turtle.down()
turtle.color('#aa0000')
turtle.circle(80,90)
turtle.up()
#
turtle.right(135)
x=0
y=50
turtle.goto(x,y)
turtle.width(2)
turtle.color('black')
turtle.down()
turtle.forward(100)
#
turtle.mainloop()

```

Для того, чтобы изобразить улыбку, потребовалось после перемещения пера в начальную точку дуги (левую) повернуть перо на 45 градусов. Дело в том, что изначально направлением «вперёд» для пера является направление вправо (как показано на рис. 3.1). Окружности и дуги рисуются как касательные к этому «вектору», начинаясь в точке с текущими координатами пера. Поэтому для улыбки потребовалось изменить направление «вектора».

Далее, перо, первоначально сориентированное на 45 градусов вправо, после прохождения дуги в 90 градусов соответственно изменило своё направление. Поэтому для получения вертикальной линии его пришлось дополнительно повернуть.

Можно поэкспериментировать с рисованием домиков, солнышка и более сложных композиций. Однако для формирования сложных кривых (например,

графиков функций) с помощью этого модуля придётся многократно выполнять команду `goto(x,y)`. В этом легко убедиться, попытавшись нарисовать, например, график параболы.

3.1.1 Задания и упражнения

1. Как в примерах кода, формирующего изображения на рис. 3.2 и 3.3, применить кортежи?
2. Напишите код для создания изображения «домика» (квадрат под треугольником) без подъёма пера при условии однократного перемещения по каждой линии.
3. Рассчитайте координаты и напишите код для создания изображения «солнца» (круг и расходящиеся от него отрезки) так, чтобы «лучи» начинались на расстоянии 2 точки от круга (не менее 8-ми лучей).
4. Напишите код для построения графика степенной функции ($y = ax^b$) с началом координат в левой нижней четверти окна рисования так, чтобы кривая проходила практически через всё окно.

3.2 Пользовательские подпрограммы и моделирование. Модуль Tkinter

Гораздо более серьёзными возможностями, чем модуль `turtle`, обладает модуль `Tkinter`. Основное предназначение этого модуля — создание графических интерфейсов (GUI — Graphical User Interface) для программ на Python. Но благодаря наличию элемента графического интерфейса (или, как говорят, «виджета») `canvas` («холст») `Tkinter` можно применять для рисования на основании координат, рассчитанных по формулам, используя доступные элементы векторной графики — кривые, дуги, эллипсы, прямоугольники и пр.

Рассмотрим задачу построения графика некоторой функции по вычисляемым точкам с помощью `Tkinter`.

Поскольку `Tkinter` позволяет работать с элементами GUI, создадим окно заданного размера, установим для него заголовок и цвет фона «холста», а также снабдим окно программной «кнопкой». На «холсте» определим систему координат и нарисуем «косинусоиду».

```
# -*- coding: utf-8 -*-  
import Tkinter  
import math  
#  
tk=Tkinter.Tk()  
tk.title("Sample")  
#
```

```
button=Tkinter.Button(tk)
button["text"]="Закрыть"
button["command"]=tk.quit
button.pack()
#
canvas=Tkinter.Canvas(tk)
canvas["height"]=360
canvas["width"]=480
canvas["background"]="#eeeeff"
canvas["borderwidth"]=2
canvas.pack()
#
canvas.create_text(20,10,text="20,10")
canvas.create_text(460,350,text="460,350")
#
points=[]
ay=150
y0=150
x0=50
x1=470
dx=10
#
for n in range(x0,x1,dx):
    y=y0-ay*math.cos(n*dx)
    pp=(n,y)
    points.append(pp)
#
canvas.create_line(points,fill="blue",smooth=1)
#
y_ave=[]
yy=(x0,0)
y_ave.append(yy)
yy=(x0,y0+ay)
y_ave.append(yy)
canvas.create_line(y_ave,fill="black",width=2)
#
x_ave=[]
xx=(x0,y0)
x_ave.append(xx)
xx=(x1,y0)
x_ave.append(xx)
canvas.create_line(x_ave,fill="black",width=2)
#
tk.mainloop()
```

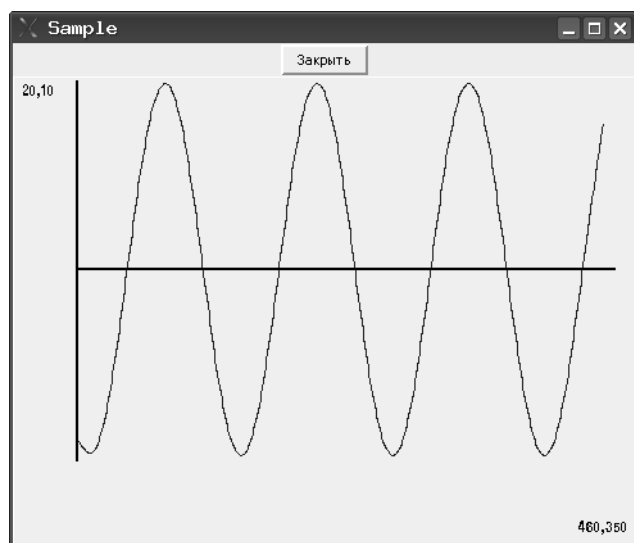


Рис. 3.4: Окно Tkinter с кнопкой и графиком

Посмотрим на результат (рис. 3.4), и разберём текст примера.

Итак, первые три строки программы понятны — устанавливается кодовая страница и подключаются библиотеки. Поскольку в примере должны использоваться тригонометрические функции, необходимо подключить модуль `math`.

Затем создаётся так называемое «корневое» окно — говоря научным языком, «экземпляр интерфейсного объекта Tk», который представляет собой просто окно без содержимого. Для этого окна устанавливается значение свойства `title` (создаётся заголовок).

Далее начинается заполнение окна интерфейсными объектами («виджетами» — `widgets`). В данном примере используется два объекта — кнопка и «холст». Для размещения объекта в окне используется метод `pack()`, а порядок объектов определяется порядком выполнения этой функции.

Кнопка создаётся как экземпляр объекта `Button` модуля Tkinter, связанный с «корневым» окном. Для кнопки можно установить текст надписи (свойство `text`) и связать кнопку с выполнением какой-либо команды (функции или процедуры), установив значение свойства `command`.

В приведённом примере кнопка связана с командой закрытия окна и прекращения работы интерпретатора, однако ничто не мешает также закрывать окно нашего «приложения» обычным образом — с помощью стандартной кнопки закрытия окна в верхнем правом углу.

После создания и размещения кнопки создаётся холст. Для элемента (объекта) `canvas` указываются высота, ширина, цвет фона и отступ от границ окна (таким образом, размеры окна получаются несколько больше, чем размеры объекта `canvas`). Размеры окна автоматически подстраиваются так, чтобы обеспечить размещение всех объектов (элементов интерфейса).

Прежде чем приступить к рисованию, исследуем систему координат. Поскольку размеры окна уже нами заданы, полезно определить, где находится точка с координатами $(0, 0)$. Как видно из попыток вывести значения координат с помощью метода `canvas.create_text()`, начало координат находится в верхнем левом углу холста.

Теперь, определившись с координатами, можно выбрать масштабные коэффициенты и сдвиги и сформировать координаты точек для рисования кривой.

При использовании метода `canvas.create_line()` в качестве координат требуется список пар точек (кортежей) (x, y) . Этот список формируется в цикле с шагом dx .

Для линии графика устанавливаются цвет и режим сглаживания. Сглаживание обеспечивает некоторую «плавность» кривой. Если его убрать, линия будет состоять из отрезков прямых. Кроме того, для линий можно устанавливать толщину, как это показано на примере осей координат.

3.2.1 Моделирование математических функций

Пусть требуется построить график функции, выбираемой из заданного списка. Здесь потребуется уже использование дополнительных интерфейсных элементов модуля Tkinter, а также создание собственных (пользовательских) процедур или функций для облегчения понимания кода.

Результат решения задачи (вариант внешнего вида «приложения») показан на рис. 3.5.

Для выбора вида математической функции используется раскрывающийся список, после выбора вида функции и нажатия на кнопку «Нарисовать» на «холсте» схематически изображается график этой функции. Кнопка «Закрыть» закрывает наше «приложение». Теперь посмотрим на код.

```
# -*- coding: utf-8 -*-
import Tkinter
import math
#
# Пользовательские процедуры
def plot_x_ace(x0, y0, x1):
    x_ace=[]
    xx=(x0, y0)
    x_ace.append(xx)
    xx=(x1, y0)
    x_ace.append(xx)
```

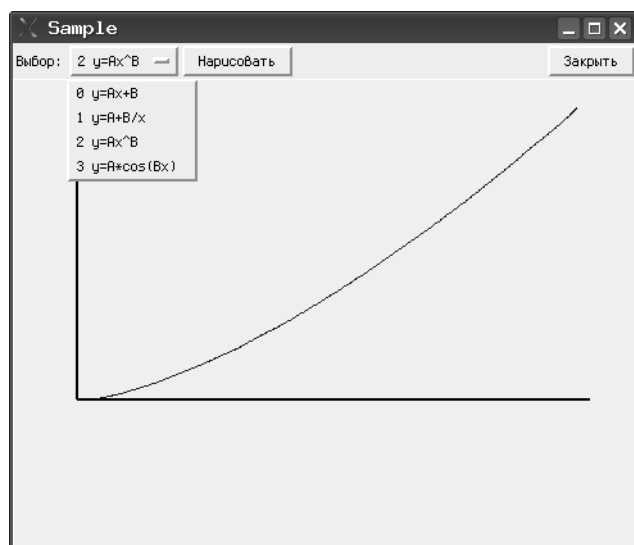


Рис. 3.5: Построение графика для функции, выбираемой из списка

```

        canvas.create_line(x_ace, fill="black", width=2)
#
def plot_y_ace(x0, y0, y1):
    y_ace=[]
    yy=(x0, y1)
    y_ace.append(yy)
    yy=(x0, y0)
    y_ace.append(yy)
    canvas.create_line(y_ace, fill="black", width=2)
#
def plot_func0(x0, x1, dx, y0, y1):
    x0i=int(x0)
    x1i=int(x1)
    y0i=int(y0)
    y1i=int(y1)
    a=y1
    b=(y0-y1)/(x1-x0)
    points=[]
    for x in range(x0i, x1i, dx):
        y=int(a+b*x)
        pp=(x, y)
        points.append(pp)
#

```

```

        canvas.create_line(points, fill="blue", smooth=1)
        plot_y_ave(x0i, y0i, y1i)
        plot_x_ave(x0i, y0i, x1i)
#
def plot_func1(x0, x1, dx, y0, y1):
    x0i=int(x0)
    x1i=int(x1)
    y0i=int(y0)
    y1i=int(y1)
    a=y0
    b=y0-y1
    points=[]
    for x in range(x0i, x1i, dx):
        y=int(a-y1i*b/x)
        pp=(x, y)
        points.append(pp)
#
    canvas.create_line(points, fill="blue", smooth=1)
    plot_y_ave(x0i, y0i, y1i)
    plot_x_ave(x0i, y0i, x1i)
#
def plot_func2(x0, x1, dx, y0, y1):
    x0i=int(x0)
    x1i=int(x1)
    y0i=int(y0)
    y1i=int(y1)
    a=(y0-y1)/(15*x1)
    b=1+((y0-y1)/(x1-x0))
    points=[]
    for x in range(x0i, x1i, dx):
        y=y0i-int(a*(x-x0i)**b)
        pp=(x, y)
        points.append(pp)
#
    canvas.create_line(points, fill="blue", smooth=1)
    plot_y_ave(x0i, y0i, y1i)
    plot_x_ave(x0i, y0i, x1i)
#
def plot_func3(x0, x1, dx, y0, y1):
    x0i=int(x0)
    x1i=int(x1)
    y0i=int(y0)
    y1i=int(y1)
    ay=150

```

```

y0i=150
points=[]
for x in range(x0i,x1i,dx):
    y=y0i-ay*math.cos(x*dx)
    pp=(x,y)
    points.append(pp)

#
canvas.create_line(points,fill="blue",smooth=1)
plot_y_ave(x0i,0,y0i+ay)
plot_x_ave(x0i,y0i,x1i)

#
def DrawGraph():
    fn=func.get()
    f=fn[0]
    x0=50.0
    y0=250.0
    x1=450.0
    y1=50.0
    dx=10
    #
    if f=="0":
        canvas.delete("all")
        plot_func0(x0,x1,dx,y0,y1)
    elif f=="1":
        canvas.delete("all")
        plot_func1(x0,x1,dx,y0,y1)
    elif f=="2":
        canvas.delete("all")
        plot_func2(x0,x1,dx,y0,y1)
    else:
        canvas.delete("all")
        plot_func3(x0,x1,dx,y0,y1)

#
# Основная часть
tk=Tkinter.Tk()
tk.title("Sample")
# Верхняя часть окна со списком и кнопками
menuframe=Tkinter.Frame(tk)
menuframe.pack({"side":"top","fill":"x"})
# Надпись для списка
lbl=Tkinter.Label(menuframe)
lbl["text"]="Выбор:"
lbl.pack({"side":"left"})
# Инициализация и формирование списка

```



```

func=Tkinter.StringVar(tk)
func.set('0_y=Ax+B')
#
fspis=Tkinter.OptionMenu(menuframe, func,
    '0_y=Ax+B',
    '1_y=A+B/x',
    '2_y=Ax^B',
    '3_y=A*cos(Bx)')
fspis.pack({"side": "left"})
# Кнопка управления рисованием
btnOk=Tkinter.Button(menuframe)
btnOk["text"]="Нарисовать"
btnOk["command"]=DrawGraph
btnOk.pack({"side": "left"})
# Кнопка закрытия приложения
button=Tkinter.Button(menuframe)
button["text"]="Закрыть"
button["command"]=tk.quit
button.pack({"side": "right"})
# Область рисования (холст)
canvas=Tkinter.Canvas(tk)
canvas["height"]=360
canvas["width"]=480
canvas["background"]="#eeeeff"
canvas["borderwidth"]=2
canvas.pack({"side": "bottom"})
tk.mainloop()

```

Основная часть программы (интерфейсная) начинается с момента создания корневого окна (инструкция `tk=Tkinter.Tk()`). В этом окне располагаются два интерфейсных элемента — рамка (Frame) и холст (canvas). Рамка является «контейнером» для остальных интерфейсных элементов — текстовой надписи (метки — Label), раскрывающегося списка вариантов (OptionMenu) и двух кнопок. Как видно, кнопка закрытия стала объектом рамки, а не корневого окна, но по-прежнему закрывает всё окно.

Для получения нужного расположения элементов метод `pack()` использует с указанием, как именно размещать элементы интерфейса (к какой стороне элемента-контейнера их нужно «прижимать»).

Есть некоторые тонкости в создании раскрывающегося списка. Для успешного выполнения этой операции нужно предварительно сформировать строку (а точнее, объект `Tkinter.StringVar()`) и определить для этого объекта значение по умолчанию (это значение будет показано в только что запущенном приложении). Затем при определении объекта `OptionMenu()` список значений дополняется. При

выборе элемента списка изменяется значение именно этой строки и для дальнейшей работы нужно его анализировать, что и делается в процедуре `DrawGraph()`.

«Вычислительная» часть, а именно, все процедуры и функции, обеспечивающие вычисления координат точек и рисование линий, вынесена в начало текста программы.

Определение каждой пользовательской подпрограммы обеспечивается составным оператором **def**. Поскольку эти подпрограммы занимаются только рисованием, они не возвращают никаких значений (т.е. результаты выполнения этих подпрограмм не присваиваются никаким переменным).

Собственно подпрограмма рисования графика `DrawGraph()` вызывается при нажатии кнопки «Нарисовать», и имя этой подпрограммы является командой, которая сопоставляется кнопке.

Эта подпрограмма берёт значение из списка (метод `get()`), выбирает первый символ получившейся строки и в зависимости от этого символа вызывает другие подпрограммы для построения конкретных графиков с установленными масштабными коэффициентами.

Перед рисованием следующего графика математической функции холст очищается командой `canvas.delete("all")`.

Для построения графика каждой функции вычисляются собственные масштабные коэффициенты, поэтому их вычисление включено в код соответствующей подпрограммы. Кроме того, для графика нужны целые значения координат, поэтому в каждой подпрограмме выполняются соответствующие преобразования с помощью функции `int()`.

Для каждого графика требуется нарисовать оси, и действия по рисованию осей также вынесены в отдельные подпрограммы.

Таким образом, оказывается, что программу нужно читать «с конца», и писать тоже.

3.2.2 Моделирование физического явления: тело, брошенное под углом к горизонту

Теперь создадим небольшое приложение для моделирования движения тела, брошенного под углом к горизонту. Физическая задача формулируется следующим образом:

Тело брошено под углом α к горизонту с начальной высоты $h_0 = 0$ и с начальной скоростью v_0 . Определить величину угла α , при которой дальность полёта тела будет максимальной. Сопротивлением воздуха пренебречь.

Основные обозначения для решения задачи показаны на рис. 3.6.

Напишем формулы, по которым определяются координаты тела x и y в зависимости от времени.

$$x(t) = v_{\text{гор}} \cdot t = v_0 \cdot \cos(\alpha) \cdot t \quad (3.1)$$

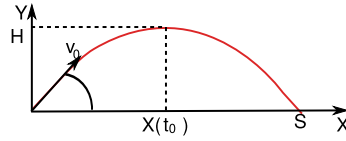


Рис. 3.6: Обозначения для задачи о теле, брошенном под углом к горизонту

$$y(t) = v_{\text{верт}} \cdot t = v_0 \cdot \sin(\alpha) \cdot t - g \cdot t^2 \quad (3.2)$$

Выразив время через координату x (на основании формулы 3.1)

$$t = \frac{x}{v_0 \cdot \cos(\alpha)} \quad (3.3)$$

и подставив выражение для времени в формулу для координаты y , получим уравнение траектории $y(x)$:

$$y(x) = x \cdot \tg(\alpha) - x^2 \cdot \frac{g}{2 \cdot v_0^2 \cdot \cos^2(\alpha)} \quad (3.4)$$

Поскольку сопротивление при движении тела отсутствует, горизонтальная составляющая скорости изменяться не будет, а изменение вертикальной составляющей определяется влиянием ускорения свободного падения.

$$v_{\text{гор}}(t) = v_0 \cdot \cos(\alpha) \quad (3.5)$$

$$v_{\text{верт}}(t) = v_0 \cdot \sin(\alpha) - g \cdot t \quad (3.6)$$

Время t_0 , через которое будет достигнута наивысшая точка траектории, найдём из условия $v_{\text{верт}} = 0$.

$$t_0 = \frac{v_0 \cdot \sin(\alpha)}{g} \quad (3.7)$$

Максимальную высоту подъёма H найдём из уравнения вертикального движения (формула 3.2) в момент времени t_0 .

$$H = y(t_0) = \frac{v_0^2 \cdot \sin^2(\alpha)}{2 \cdot g} \quad (3.8)$$

Полное время полёта T очевидно, равно $2t_0$, поэтому дальность полёта S определим как

$$S = v_{\text{гор}} \cdot T = v_0 \cdot \cos(\alpha) \cdot 2 \cdot t_0 = \frac{v_0^2 \cdot \sin(2 \cdot \alpha)}{g} \quad (3.9)$$

Все эти формулы понадобятся для вычисления координат точек траектории и параметров траектории при моделировании.

Текст программы с пользовательскими подпрограммами приведён ниже.

```

# -*- coding: utf-8 -*-
# Моделирование задачи о теле
# брошенном под углом к горизонту
#
import Tkinter
import math
#
def plot_x_ave(x0, y0, x1):
    x_ave=[]
    xx=(x0, y0)
    x_ave.append(xx)
    xx=(x1, y0)
    x_ave.append(xx)
    canvas.create_line(x_ave, fill="black", width=2)
#
def plot_y_ave(x0, y0, y1):
    y_ave=[]
    yy=(x0, y1)
    y_ave.append(yy)
    yy=(x0, y0)
    y_ave.append(yy)
    canvas.create_line(y_ave, fill="black", width=2)
#
def DrawGraph():
# Получаем и пересчитываем параметры
    dta=sc.get()
    alpha=dta*math.pi/180
    dtlbl=clist.get()
# Очищаем область для текста
    canvas.create_rectangle(x1i-90, y1i-50,
        x1i+50, y1i+10, fill="#eeeeff")
# Считаем g=10, v0 подбираем, чтобы всё влезало в canvas
    g=10.0
    v0=63
#
    S=int(((v0**2)*math.sin(2*alpha))/g)
    H=int(((v0**2)*(math.sin(alpha)**2))/(2*g))
#
    points=[]
    for x in range(x0i, x1i):
        xx=(x-x0)

        y=(xx*math.tan(alpha))-((xx**2)*g/\
            (2*(v0**2)*(math.cos(alpha)**2)))

```

```

#
    if y > 0:
        yy=int(y0-y)
    else:
        yy=y0i
#
    pp=(x, yy)
    points.append(pp)
# Собственно график
    canvas.create_line(points, fill=dtlbl, smooth=1)
    plot_x_ace(x0i, y0i, x1i)
# Параметры графика
    dtext="Дальность: " +str(S)
    vtext="Высота: " +str(H)
    dalnost=canvas.create_text(x1i-70, y1i-30, text=dtext,
        fill=dtlbl, anchor="w")
    vysota=canvas.create_text(x1i-70, y1i-10, text=vtext,
        fill=dtlbl, anchor="w")
#
# Основная часть
tk=Tkinter.Tk()
tk.title("Моделирование полёта")
# Верхняя часть окна со списком и кнопками
menuframe=Tkinter.Frame(tk)
menuframe.pack({"side": "top", "fill": "x"})
# Надпись для списка
lbl=Tkinter.Label(menuframe)
lbl["text"]="Выбор цвета: "
lbl.pack({"side": "left"})
# Инициализация и формирование списка
clist=Tkinter.StringVar(tk)
clist.set('black')
#
cspis=Tkinter.OptionMenu(menuframe, clist,
    'red',
    'green',
    'blue',
    'cyan',
    'magenta',
    'purple',
    'black')
cspis.pack({"side": "left"})
# Кнопка управления рисованием
btnOk=Tkinter.Button(menuframe)

```

```
btnOk["text"]="Нарисовать"
btnOk["command"]=DrawGraph
btnOk.pack({"side":"left"})
# Кнопка закрытия приложения
button=Tkinter.Button(menuframe)
button["text"]="Закрыть"
button["command"]=tk.quit
button.pack({"side":"right"})
#
# Надпись для шкалы углов
lbl2=Tkinter.Label(tk)
lbl2["text"]="Угол, градусы: "
lbl2.pack({"side":"top"})
# Шкала углов
sc=Tkinter.Scale(tk,from_=0,to=90,orient="horizontal")
sc.pack({"side":"top","fill":"x"})
#
# Область рисования (холст)
canvas=Tkinter.Canvas(tk)
canvas["height"]=360
canvas["width"]=480
canvas["background"]="#eeeeff"
canvas["borderwidth"]=2
canvas.pack({"side":"bottom"})
#
# Установки осей координат
x0=50.0
y0=300.0
x1=450.0
y1=50.0
#
x0i=int(x0)
x1i=int(x1)
y0i=int(y0)
y1i=int(y1)
# Оси координат
plot_x_ave(x0i,y0i,x1i)
plot_y_ave(x0i,y0i,y1i)
#
tk.mainloop()
```

Результат работы с моделью показан на рис. 3.7.

Реализация модели имеет ряд особенностей. Во-первых, величина ускорения свободного падения g принята как 10. Во-вторых, модуль начальной скорости вы-

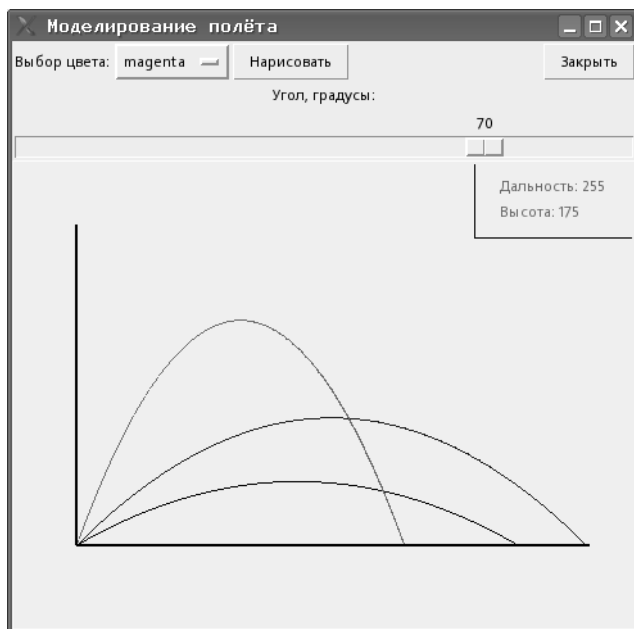


Рис. 3.7: Поиск угла для достижения максимальной дальности на модели

бран так, чтобы при любых значениях угла вся траектория попадала в область графика. Не совсем правильно с точки зрения принципа разделения программ и данных установка значений для g и v_0 прямо в коде, но такое решение значительно упрощает работу с моделью.

«Ползунок» на шкале установки углов показывает значения в градусах, а для правильных вычислений в тригонометрических функциях эти значения нужно перевести в радианы.

Высота и дальность полёта пишутся для каждой траектории соответствующим цветом в прямоугольнике в верхнем правом углу. Для каждой следующей траектории этот прямоугольник рисуется заново и текст переписывается.

В этой главе рассмотрены лишь некоторые базовые возможности модуля Tkinter и использования Python для создания моделей. Если возникнет желание более подробно познакомиться с применением объектов и методов этого модуля, можно изучить оригинальную документацию и другие примеры, используя ресурсы Интернет.

3.2.3 Задачи и упражнения

1. Для примера, показанного на рис. 3.4, нанесите на оси метки и проставьте значения в масштабе графика.

2. Напишите подпрограмму формирования строки со значениями координат для примера, показанного на рис. 3.2.
3. Напишите подпрограммы для нанесения меток и вывода значений по горизонтальной и вертикальной осям для примера моделирования математических функций.
4. В модели тела, брошенного под углом α к горизонту, напишите подпрограммы вывода метки «точки падения» и метки максимальной высоты для каждой траектории.
5. Модифицируйте код для моделирования полёта так, чтобы можно было изменять начальную скорость, а график автоматически масштабировался в области рисования.

Глава 4

Методические указания для учителей и преподавателей

4.1 Введение. Почему Python?

Выбор в пользу того или иного языка программирования является следствием огромного количества факторов — от требований эффективного использования ресурсов вычислительной системы до наличия в нужное время подходящей книжки.

Поэтому, чтобы избежать непродуктивного и спорного сравнения различных языков программирования друг с другом (к тому же, такое сравнение провести крайне трудно ввиду их огромного количества и разнообразия параметров сравнения), рассмотрим только аргументы в пользу выбора языка Python (общепринятое произношение — «Питон», хотя допускается и «Пайтон»).

1. Python — сравнительно «молодой» язык. Создавая его в 1990—1991 годах, его автор Гвидо ван Россум (Guido van Rossum) учёл все достоинства и недостатки предшествующих языков программирования.
2. Python имеет достаточно долгую историю развития и использования (почти 20 лет). В настоящее время Python поддерживается обширным международным сообществом разработчиков.
3. Python — развивающийся язык, используемый в реальных проектах. Это означает, что его изучение не пройдёт напрасно.
4. Средства для работы с Python относятся к категории свободно распространяемого программного обеспечения (СПО). Это гарантирует во-первых, от каких-либо претензий относительно использования «интеллектуальной собственности», а во-вторых, от превращения Python в обозримом будущем в «мёртвый» язык (вспомните «популярный» Turbo Pascal).

5. Python имеет обширную область применения. Так, на Python создаются расширения к графическому редактору GIMP, на Python можно программировать в офисном пакете OpenOffice.org, на Python пишутся сценарии для пакета 3D-моделирования Blender, на Python написаны системы управления контентом Plone и MoinMoin Wiki, Python активно используется при создании компьютерных игр.
6. Python — интерпретируемый язык, что очень удобно при обучении программированию. Интерпретатор Python входит в большинство дистрибутивов GNU/Linux (и разумеется, в ПСПО для школ).
7. Существует множество средств, облегчающих процесс создания программ на Python. Это и специализированные лексические анализаторы, и редакторы для программистов (например, Kate и Bluefish), и интегрированные среды разработки (IDE).
8. Наконец, многие средства для работы с Python являются кросс-платформенными, а в конструкциях языка поддерживаются многобайтные кодировки (Unicode), поэтому программы на Python легко переносятся с одной среды функционирования на другую.

4.2 Требования к программной конфигурации

Для успешного проведения практикума по алгоритмизации и программированию на Python на рабочих местах должны быть установлены собственно Python (версия не ниже 2.4), модули Tkinter и NumPy, среды разработки на Python — IDLE, Eric или Geany, а также какие-либо эмуляторы терминалов — xterm, rxvt и т.п.

В сборке от ALT Linux следует проверить наличие в системе следующих пакетов

- geany
- eric
- xterm
- python
- python-base
- python-doc
- python-module-numpy
- python-modules
- python-modules-encodings
- python-modules-tkinter
- python-tools-idle

Некоторые из перечисленных пакетов будут установлены по зависимостям при установке Python, Eric и Geany с помощью менеджера пакетов, остальные нужно установить «вручную».

При создании программ (этот процесс обозначается звучным словом «разработка») удобно одновременно видеть текст программы и результаты её выполнения. Хорошо также, если при этом по-разному выделяются ключевые слова, названия функций и их аргументы, а также сразу же показываются строки, содержащие ошибки. Кроме того, бывает полезно выполнять программу по шагам и при этом следить за значениями каких-то переменных. Все эти возможности реализуются в так называемых *интегрированных средах разработки* (Integrated Development Environment, IDE).

Современные IDE, входящие в дистрибутивы Linux, могут работать с разными языками программирования. Существует IDE, лучше всего приспособленные для работы с одним конкретным языком, которые с другими языками работают, так сказать, факультативно. Кроме того, существуют IDE, которые одинаково успешно обеспечивают работу с самыми разными языками, как в режиме интерпретатора, так и в режиме компилятора.

В зависимости от версии ALT Linux удобно пользоваться либо Geany, либо Eric. Далее будут рассмотрены особенности работы в обоих IDE.

4.3 Основные понятия и определения (гlossарий)

В дальнейшем часто придётся использовать термины, связанные с процессом разработки и функционирования программ, поэтому здесь приведён краткий словарь, чтобы уже не возвращаться к проблеме определений. Для составления этого словаря использованы в основном материалы сайта «Гlossарий.Ру» (<http://glossary.ru>).

4.3.0.1 Алгоритм

Алгоритм — точное предписание исполнителю совершить определённую последовательность действий для достижения поставленной цели за конечное число шагов.

4.3.0.2 Данные

Данные — сведения:

- полученные путём измерения, наблюдения, логических или арифметических операций; и
- представленные в форме, пригодной для постоянного хранения, передачи и (автоматизированной) обработки.

4.3.0.3 Тип данных

Тип данных — характеристика набора данных, которая определяет:

- диапазон возможных значений данных из набора;

- допустимые операции, которые можно выполнять над этими значениями;
- способ хранения этих значений в памяти.

Различают:

- простые типы данных: целые, действительные числа, символы, строки, логические величины;
- составные типы данных: массивы, файлы и др.

4.3.0.4 Программа

Программа — согласно ГОСТ 19781-90 — данные, предназначенные для управления конкретными компонентами системы обработки информации в целях реализации определённого алгоритма.

4.3.0.5 Алгоритмический язык (язык программирования)

Язык программирования — искусственный (формальный) язык, предназначенный для записи алгоритмов. Язык программирования задаётся своим описанием и реализуется в виде специальной программы: компилятора или интерпретатора.

4.3.0.6 Транслятор языка программирования

Транслятор — в широком смысле — программа, преобразующая текст, написанный на одном языке, в текст на другом языке.

Транслятор — в узком смысле — программа, преобразующая: программу, написанную на одном (входном) языке в программу, представленную на другом (выходном) языке.

Транслятор языка программирования — программа, преобразующая *исходный текст* программы на языке программирования в *машинный язык* вычислительной системы, на которой эта программ должна выполняться.

4.3.0.7 Интерпретатор

Интерпретатор — транслятор, способный параллельно переводить и выполнять программу, написанную на алгоритмическом языке высокого уровня.

4.3.0.8 Компилятор

Компилятор — программа, преобразующая текст, написанный на алгоритмическом языке, в программу, состоящую из машинных команд. Компилятор создаёт законченный вариант программы на машинном языке.

4.3.0.9 Константа

Константа — в программировании — элемент данных, который занимает место в памяти, имеет имя и определённый тип, причём его значение никогда не меняется.

4.3.0.10 Переменная

Переменная — в языках программирования — именованная часть памяти, в которую могут помещаться разные значения. Причём в каждый момент времени переменная имеет единственное значение (или единственный набор значений). В процессе выполнения программы значение переменной может изменяться.

Тип переменных определяется типом данных, которые они представляют.

4.3.0.11 Подпрограмма

Подпрограмма — самостоятельная часть программы, которая разрабатывается независимо от других частей и затем вызывается по имени.

4.3.0.12 Функция

Подпрограмма, которая на основе некоторых данных (аргументов функции) вычисляет значение некоторой переменной («функция возвращает значение»).

4.3.0.13 Объект

Понятие объектно-ориентированного программирования, программный модуль, объединяющий в единое целое данные и программы, манипулирующие данными. Объект характеризуется свойствами, которые являются параметрами объекта и методами, которые позволяют воздействовать на объект и его свойства.

4.3.0.14 Метод

Действие в виде процедуры, которое выполняется объектом (иногда говорят — выполняется над объектом).

4.3.0.15 Идентификатор

Идентификатор — символьное имя переменной или подпрограммы, которые однозначно идентифицируют их в программе.

4.3.0.16 Выражение

Выражение — конструкция на языке программирования, предназначенная для выполнения вычислений. Выражение состоит из операндов, объединённых

знаками операций. Различают арифметические, логические и символьные выражения.

4.3.0.17 Операнд

Операнд — константа, переменная, функция, выражение и другой объект языка программирования, над которым производятся операции.

4.3.0.18 Арифметическая операция

Арифметическая операция — вычислительная операция над числами.

Во многих языках программирования определены двуместные арифметические операции: сложения, вычитания, умножения, деления, деления нацело, вычисление остатка от деления.

4.3.0.19 Логическая операция

Логическая операция — операция над логическими («булевыми») операндами, принимающими значения «Истина» или «Ложь». Наиболее распространёнными являются следующие операции:

- многоместное логическое сложение;
- многоместное логическое умножение;
- одноместное логическое отрицание.

(«Многоместная» операция означает, что в ней может два и более операндов, а в «одноместной» или «унарной» операции участвует только один операнд).

4.3.0.20 Операция отношения

Операция отношения производит сравнение двух величин. Результат операции отношения является «булевой» переменной, принимающей значение «Истина» (True или логическая 1) или «Ложь» (False или логический 0).

4.3.0.21 Массив (массив данных)

Совокупность, как правило, однотипных данных, каждое из которых идентифицируется с именем массива и индексом (индексами).

В зависимости от количества индексов массивы бывают одномерные (линейные), двумерные и т.д.

4.3.0.22 Индекс

Номер (или номера, если массив данных многомерный), добавляемый к имени массива, чтобы идентифицировать каждый элемент данного массива.

Например, $a[1, 3]$ означает, что определён элемент двухмерного массива a с индексом 1,3 (строка — 1, столбец — 3).

4.3.0.23 Присваивание

Операция записи значения в переменную. Во многих языках программирования определён *оператор присваивания*. Если в переменную записывается новое значение, старое стирается.

4.3.0.24 Цикл

Цикл (циклические вычисления) означают многократное выполнение одних и тех же операций. В зависимости от задачи различаются циклы с переменной (со счётчиком, с известным количеством повторений) и циклы с условием (цикл повторяется, пока не выполнится условие завершения цикла).

4.3.0.25 Зацикливание

Для циклов с условием — ситуация, при которой условие завершения цикла никогда не выполняется.

4.4 Использование IDE Geany

Один из вариантов IDE для работы с Python — кросс-платформенный пакет Geany. В программном меню Geany обычно находится в разделе «Разработка» (или «Разработка/Прочее»). Внешний вид окна Geany при первом запуске показан на рис. 4.1.

Центральная часть окна предназначена для размещения вкладок с текстами программ, в нижней части размещается несколько вкладок. Некоторые из этих вкладок будут полезны в дальнейшем, а некоторые при работе с Python (например, вкладка «Компилятор») не нужны. В панели инструментов часть кнопок недоступна (кнопки имеют серый цвет). Пока нет текста программы, для этих кнопок «нет работы».

В нижней части окна находится несколько вкладок. Вкладка «Статус» фиксирует историю открытия и закрытия файлов с текстами программ, а на вкладке «Компилятор» отображаются сообщения компилятора при компиляции исходного текста (компиляция запускается кнопкой «Скомпилировать текущий файл» в панели инструментов). Понятно, что вкладка «Компилятор» требуется только для работы с компилируемыми программами (C, Pascal и т. п.).

Вкладка «Заметки» может использоваться как своего рода «блокнот» для записи каких-то замечаний и пожеланий, а вкладка «Терминал» является встроен-

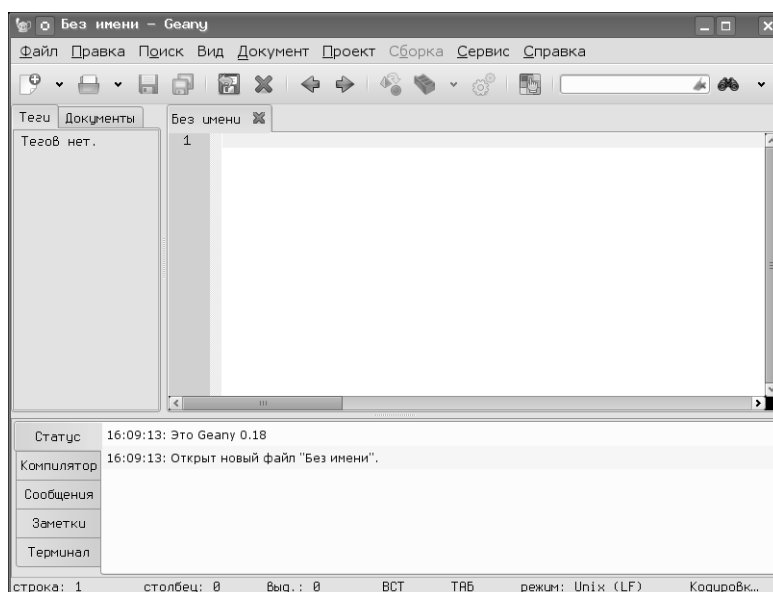


Рис. 4.1: Внешний вид IDE «Geany»

ным в Geany эмулятором терминала (Virtual terminal Emulator — VTE). Однако в дальнейшем будет использоваться «внешний эмулятор терминала» — `xterm`.

4.4.1 Первоначальная настройка

Первоначальная настройка полезна для создания комфортной «среды обитания» в IDE. Разумеется, в каждом конкретном случае важны личные предпочтения, а здесь рассмотрим параметры настройки и разумные значения параметров для работы с Python в рамках «Практикума».

Для вызова диалога настроек используется команда главного меню «Правка/Настройки» или комбинация клавиш `<CTRL>+<ALT>+<P>`. Диалог содержит несколько вкладок (рис. 4.2) и начинается всё с общих настроек.

На вкладке «Общие» диалога настройки Geany имеет смысл установить режимы «Загрузить файлы из последней сессии» и «Сохранять позицию и размеры окна». Что касается режима «Загружать виртуальный терминал» то при использовании внешнего эмулятора терминала (в нашем случае — `xterm`) он не нужен. Отключение этого режима приведёт к исчезновению вкладки «Терминал» в нижней части окна Geany, а также к исчезновению вкладки «Терминал» в диалоге настроек Geany при его последующем вызове.

Режим «Включить поддержку плагинов» нужен для автоматического вызова интерпретаторов и компиляторов языков программирования. Если для плагинов не указан дополнительный путь, Geany будет использовать пути из системы.

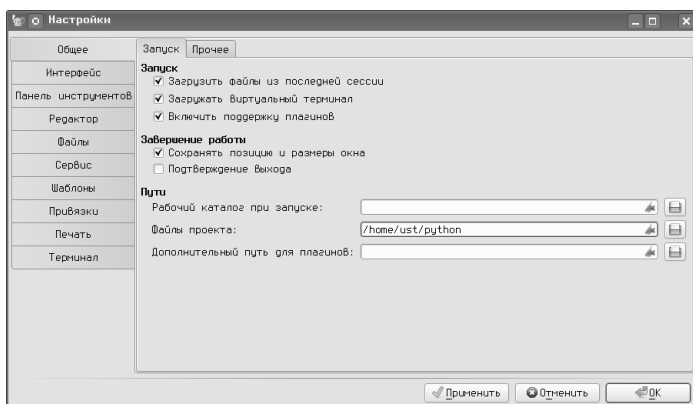


Рис. 4.2: Диалог настройки Geany — общие настройки

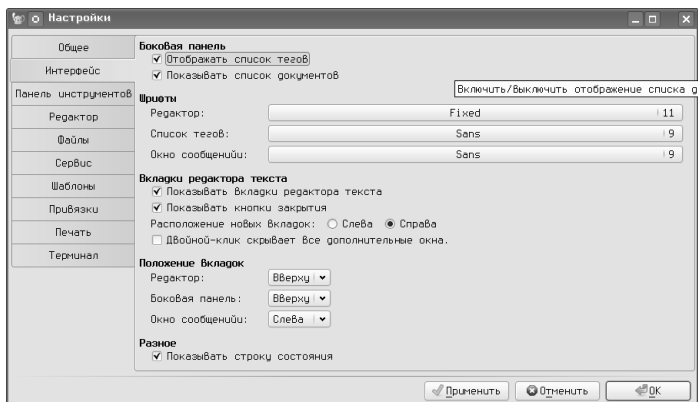


Рис. 4.3: Настройки внешнего вида IDE Geany

Рабочий каталог и каталог для файлов проекта можно установить (набрав путь в строке ввода или нажав на кнопку «Открыть папку» справа от строки ввода), но это не обязательно при включённом режиме «Загрузить файлы из последней сессии».

За внешний вид Geany отвечают три вкладки диалога настроек – «Интерфейс», «Панель инструментов» и «Редактор». На рис. 4.3 показана вкладка «Интерфейс».

На вкладке «Интерфейс» имеет смысл только настроить шрифты для области редактора и других панелей. Диалог выбора шрифтов (рис. 4.4) вызывается нажатием на кнопку с названием шрифта.

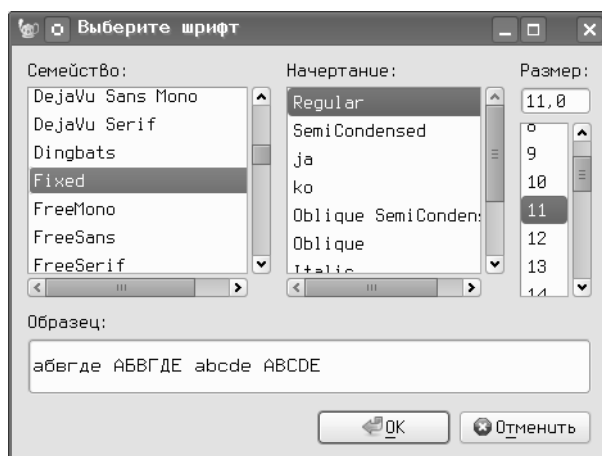


Рис. 4.4: Выбор шрифтов в Geany

Шрифты каждый выбирает для себя, в частности, автор предпочитает для редактирования программ использовать моноширинные шрифты (в именах которых содержится слово «Fixed»).

Боковая панель при использовании режимов, показанных на рис. 4.3, будет иметь две вкладки — «Теги» и «Документы». На вкладке «Документы» отображается список открытых документов, а на вкладке «Теги» при работе с языками программирования отображаются имена переменных с номерами строк, в которых эти переменные определяются и используются, а также имена использованных в программе функций с номерами строк, в которых эти функции определяются и используются.

На вкладке «Панель инструментов» (рис. 4.5), очевидно, должен быть включён режим «Показывать панель инструментов». Включение режима «Добавить панель инструментов в меню» приведёт к тому, что панель инструментов станет «продолжением» строки главного меню. В большинстве случаев это нецелесообразно, поэтому такой режим включать не рекомендуется.

Для внешнего вида панели инструментов разумно выбрать вариант «Только иконки», а размер иконок выбрать по вкусу.

Если есть желание изменить набор видимых инструментов Geany, можно использовать диалог настройки панели инструментов (кнопка «Настроить панель инструментов»). Диалог настройки показан на рис. 4.6.

В этом диалоге можно перемещать доступные элементы из левой панели в область отображаемых элементов (правую панель) кнопкой «Стрелка вправо» и убирать отображаемые элементы из панели кнопкой «Стрелка влево» (кнопки находятся в центре окна диалога). Изменение порядка кнопок, как следует из

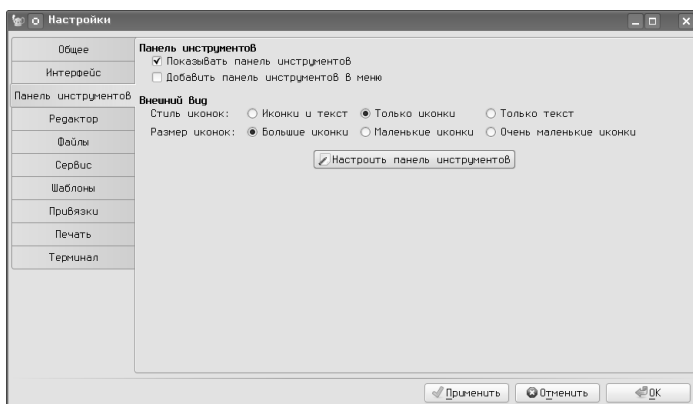


Рис. 4.5: Настройка панели инструментов в Geany

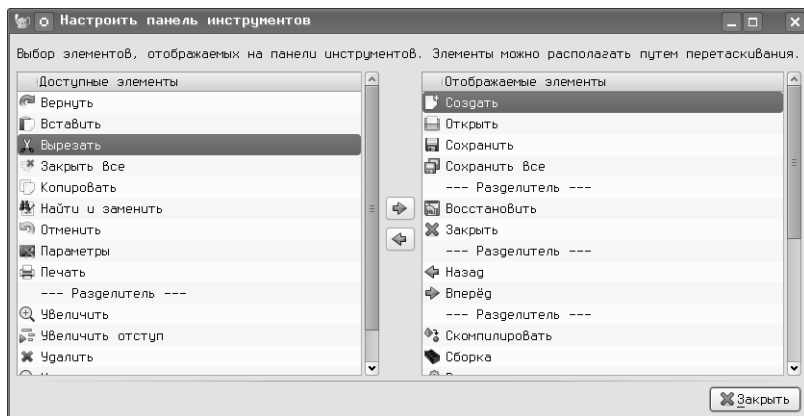


Рис. 4.6: Диалог настройки панели инструментов Geany

подсказки этого диалога, достигается путём перетаскивания выбранных элементов.

На панели инструментов Geany (рис. 4.1) имеется строка ввода слова для поиска и кнопка поиска, но отсутствуют часто используемые кнопки «Вырезать», «Копировать» и «Вставить». Поэтому при начальной настройке имеет смысл убрать строку поиска из панели и добавить нужные кнопки, расположив их примерно так, как показано на рис. 4.7.

Важно не забыть закрыть диалог настройки панели инструментов (кнопка «Закрывать» и применить сделанные изменения (кнопка «Применить» на вкладке «Панель инструментов» диалога настроек Geany).

Вид окна программы после изменений показан на рис. 4.17.

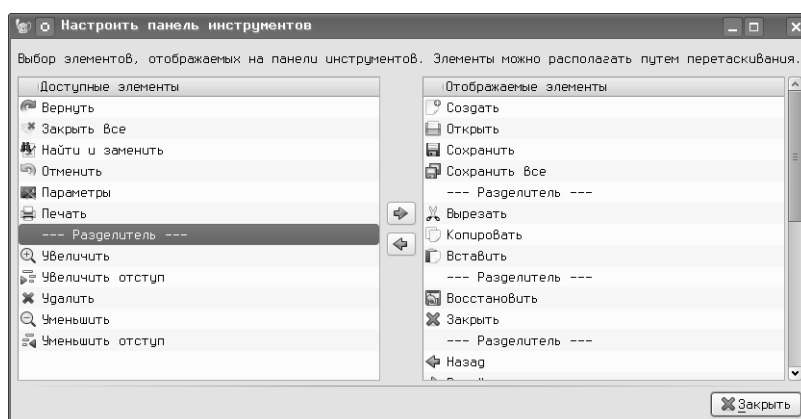


Рис. 4.7: Изменение панели инструментов Geany

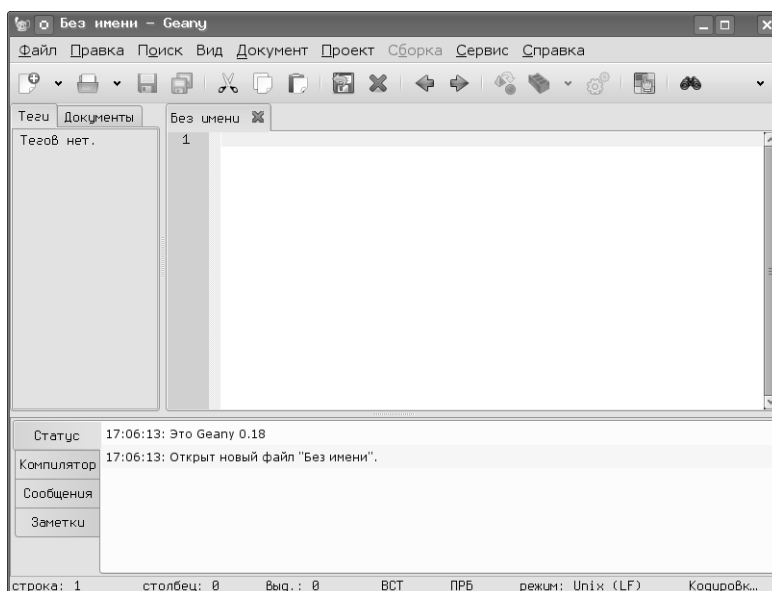


Рис. 4.8: Geany с отключённым терминалом и модифицированной панелью инструментов

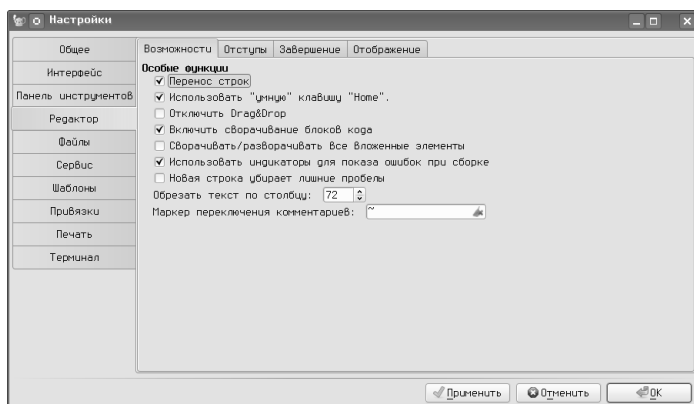


Рис. 4.9: Настройки редактора в Geany

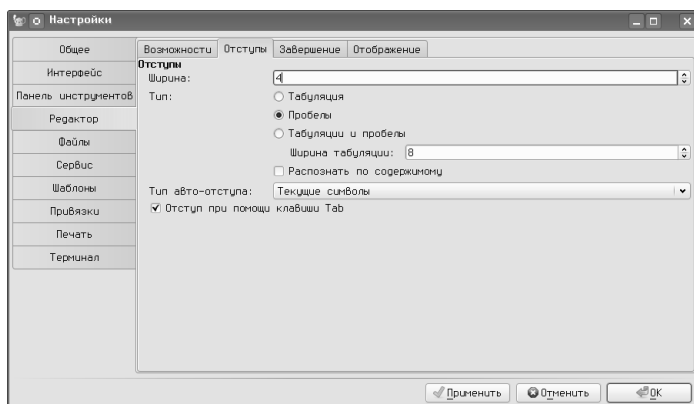


Рис. 4.10: Настройки отступов для работы с Python в Geany

Вкладка для настроек редактора (рис. 4.9) в свою очередь, является многостраничным диалогом.

На листе «Возможности» следует проследить, что включены режимы «Перенос строк» и «Включить сворачивание блоков кода». На листе «Отступы» (рис. 4.10) нужно установить ширину отступа в 4 символа (в соответствии с правилами Python) и тип отступа – «Пробелы». Режим «Отступ при помощи клавиши Tab» можно оставить без изменений.

На листе «Завершение» (рис. 4.11) при желании можно настроить режимы авто-завершения слов и автоматического создания парных скобок, однако начальные настройки являются достаточно разумными и без особой необходимости их менять не нужно.

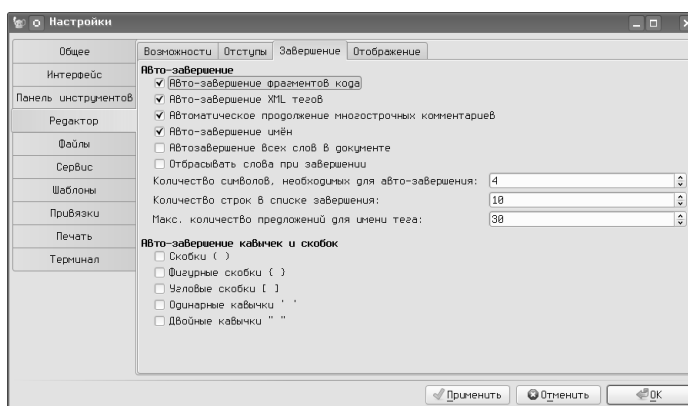


Рис. 4.11: Настройки авто-завершения в редакторе Geany

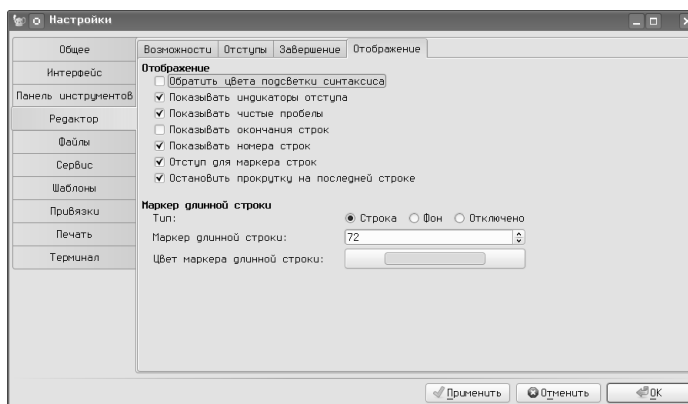


Рис. 4.12: Настройка отображения структуры программы в редакторе Geany

На листе «Отображение» в настройках редактора (рис. 4.12) нужно обязательно включить режимы «Показывать индикаторы отступа», «Показывать чистые пробелы» и «Показывать номера строк». Кнопка «Цвет маркера длинной строки» открывает диалог выбора цвета для GTK (рис. 4.13). Этим цветом будет обозначена вертикальная линия в окне редактора (условная правая граница текста).

В диалоге выбора цвета можно «крутить» треугольник по цветному кольцу, перетаскивая мышью чёрный отрезок и перетаскивать мышью чёрный кружок в пределах треугольника. Результат тут же будет показан в палитрах HSV («Тон-Насыщенность-Значение») и RGB («Красный-Зелёный-Синий»), а также в виде HTML-эквивалента («Название цвета»). И наоборот, можно в помощью

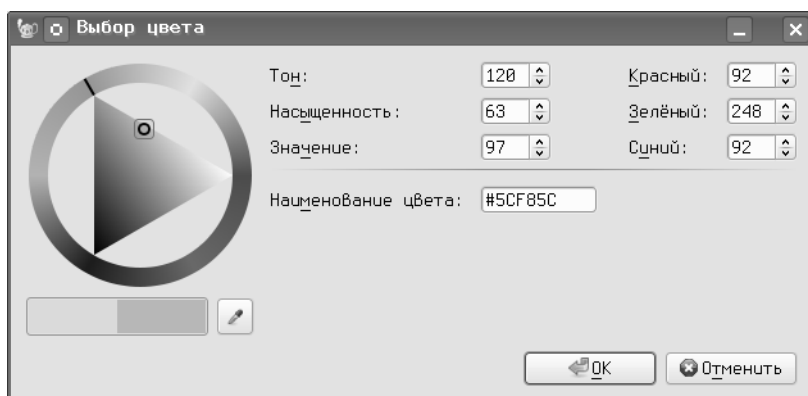


Рис. 4.13: GTK-диалог выбора цвета для обозначения правой границы текста в Geany

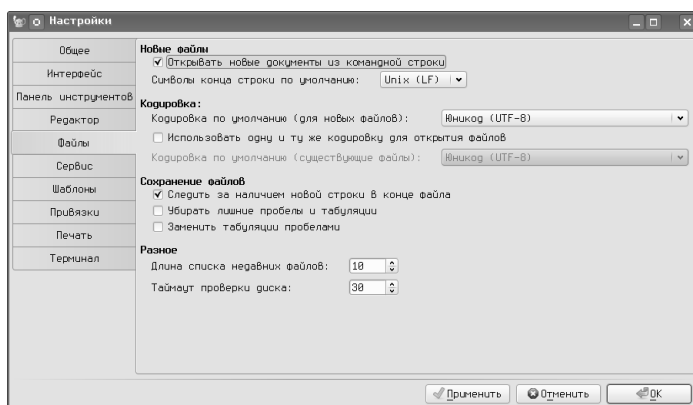


Рис. 4.14: Настройка обработки файлов в Geany

полей ввода в диалоге установить нужный цвет и он будет показан в треугольнике соответствующей ориентации. Использование кнопки «пипетка» позволяет получить цвет с любой точки экрана. Под цветным кольцом с треугольником показывается сравнение выбранного цвета с текущим.

На вкладке «Файлы» диалога настроек Geany (рис. 4.14) определяются режимы обработки файлов. Здесь можно ничего не менять, только следует обратить внимание на то, что для всех сохраняемых файлов устанавливается одинаковая кодировка. Geany корректно открывает файлы в различных кодировках, но сохраняет всегда в одной и той же.

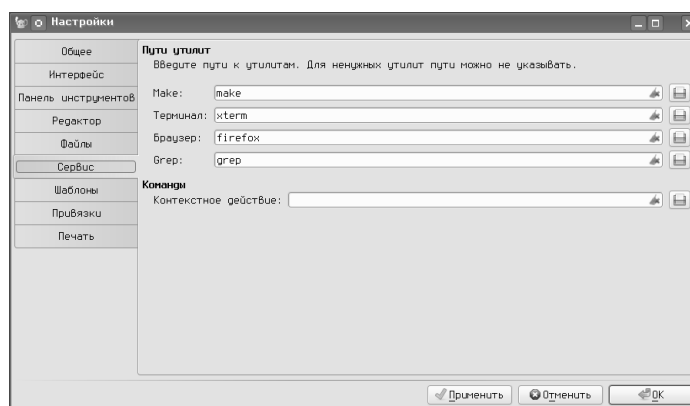


Рис. 4.15: Подключение внешних программ

На вкладке «Сервис» определяются внешние программы (утилиты), которые используются Geany для обработки исходных текстов (рис. 4.15).

Как раз здесь и определяется программа, которая будет использоваться в качестве эмулятора терминала. Как уже отмечалось выше, будет использоваться `xterm`.

Оставшиеся вкладки («Шаблоны», «Привязки» и «Печать») диалога настроек Geany пока не представляют интереса.

4.4.2 Подключение документации и её использование

Очень важно иметь возможность быстро открыть документацию. После установки пакета `python-doc` оригинальная документация находится в каталоге `/usr/share/doc/python-doc-x.y.z`, где `x.y.z` — версия Python (например, `/usr/share/doc/python-doc-2.5.2`).

Самый простой путь подключения документации — запустить любой браузер (Konqueror или Firefox), выбрать в главном меню команду «Файл/Открыть файл...» и в диалоге выбора файлов в иерархии корневого каталога найти `/usr/share/doc/python-doc-x.y.z/index.html`. После этого в браузере можно добавить этот адрес в закладки.

Общий вид документации по Python в браузере показан на рис. 4.16. Ссылки на документацию по Python на русском языке можно найти в разделе «Документация» на сайте `python.ru`. Освоение языка на основе документации из пакета `python-doc` требует от начинающих значительных усилий, поэтому на начальном этапе рекомендуется пользоваться источниками из прилагаемого к «Практикуму» списка литературы.

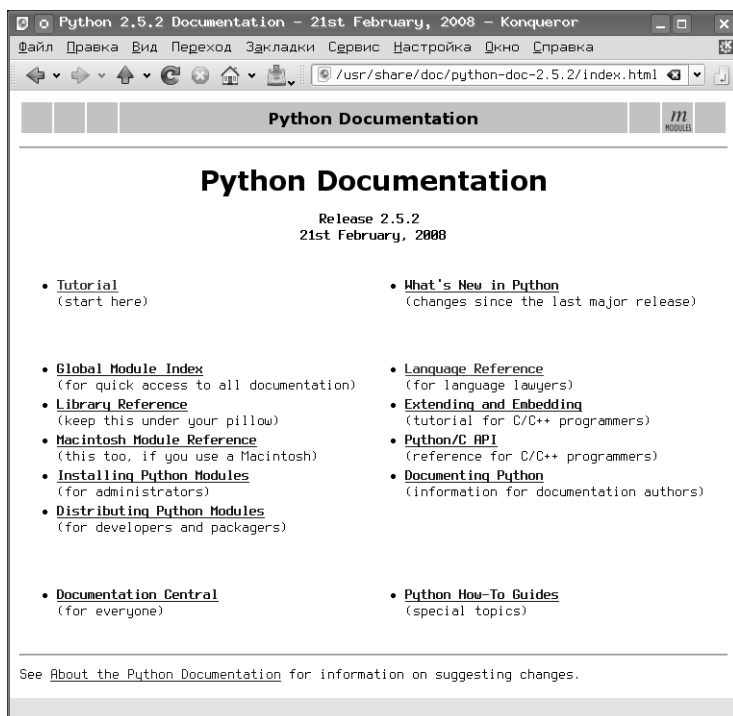


Рис. 4.16: Документация по Python (главная страница)

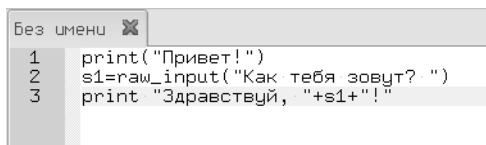


Рис. 4.17: Текст программы в окне редактора до сохранения

4.4.3 Сохранение и открытие файлов, запуск выполнения программ

Текст программы пишется в окне редактора, и пока он не сохранён, на ярлычке активной вкладки имя файла выделено красным цветом (рис. 4.17). Кроме того, для вновь набранного текста программы отсутствует подсветка синтаксиса, поскольку анализатор синтаксиса пока «не знает» языка, на котором написан этот файл.

Для сохранения файла используется команда главного меню «Файл/Сохранить» (или «Файл/Сохранить как...» при первом сохранении), что равносильно использованию комбинации клавиш <CTRL>+<S>.

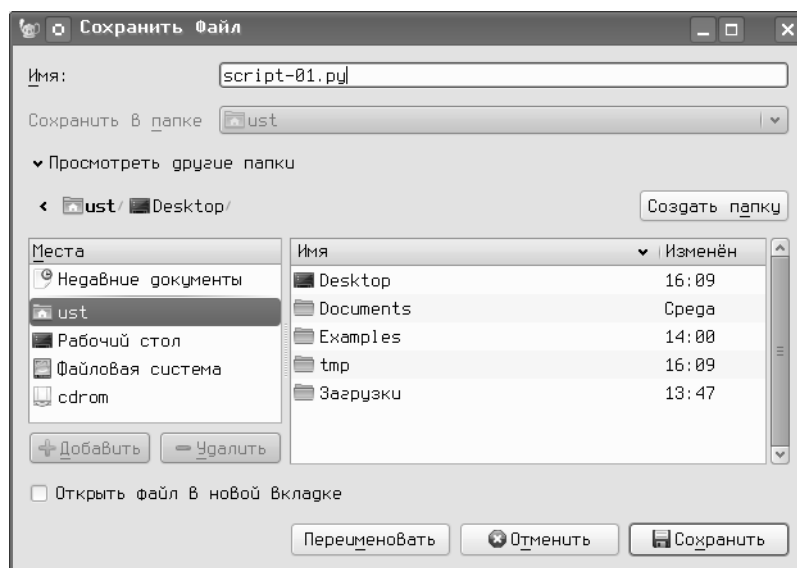


Рис. 4.18: GTK-диалог сохранения (открытия) файла

Выбор этой команды открывает GTK-диалог сохранения/открытия файла (рис. 4.18). Для получения возможности сохранения файла в каталоге по выбору пользователя нужно щёлкнуть по «стрелочке» слева от пояснения «Просмотреть другие папки». Тогда в диалоге будут показаны дополнительные панели — «Места» и «Имя». На панели «Места» приводится список наиболее часто используемых мест в файловой системе, а на панели «Имя» показан список каталогов и файлов в выбранном месте. Для переключения на новое место нужен одиночный щелчок левой кнопкой мыши в панели «Места», а для открытия папки (каталога) в панели «Имя» нужен двойной щелчок левой кнопки мыши независимо от настроек пользовательской среды в сеансе.

Если выделить каталог в панели «Имя» одиночным щелчком левой кнопкой мыши, то его можно добавить в список мест в помощью кнопки «Добавить», расположенной под панелью «Места».

При сохранении необходимо указать имя и «расширение» файла (для текстов на Python — «.py»), как показано на рис. 4.18, иначе не будет работать анализатор синтаксиса.

При нажатии на кнопку «Сохранить» закрывается диалог сохранения и в окне редактора включается подсветка синтаксиса, а также появляется список переменных в панели «Теги» (рис. 4.19). В списке тегов число в квадратных скобках справа от имени переменной означает номер строки, в которой первый раз определена эта переменная.

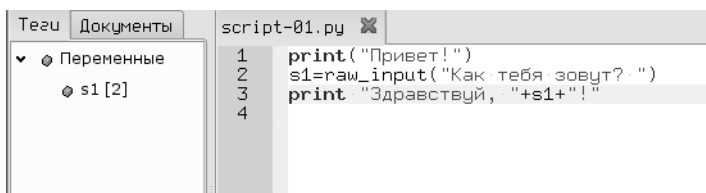


Рис. 4.19: Подсветка синтаксиса и теги для программы на Python



Рис. 4.20: Сообщение интерпретатора об ошибке, связанной с отсутствием указания кодовой страницы

Для открытия файла удобно использовать список последних файлов (команда «Файл/Недавние файлы»).

Для запуска программы на выполнение (точнее, на трансляцию и выполнение интерпретатором Python) используется клавиша <F5> или кнопка «Запустить или посмотреть текущий файл» в панели инструментов.

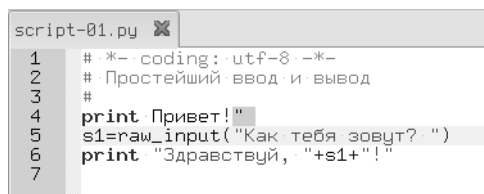
4.4.4 Обработка ошибок

Самая первая ошибка при создании программы на Python — пропуск строки с указанием кодовой страницы. Именно такая ошибка допущена в рассмотренном выше примере. На рис. 4.20 показано сообщение интерпретатора после запуска этого примера. (Закрыть окно терминала можно нажатием на <ENTER>).

В случае незакрытых кавычек или скобок Geany отмечает цветом символ, не имеющий пары (рис. 4.21).

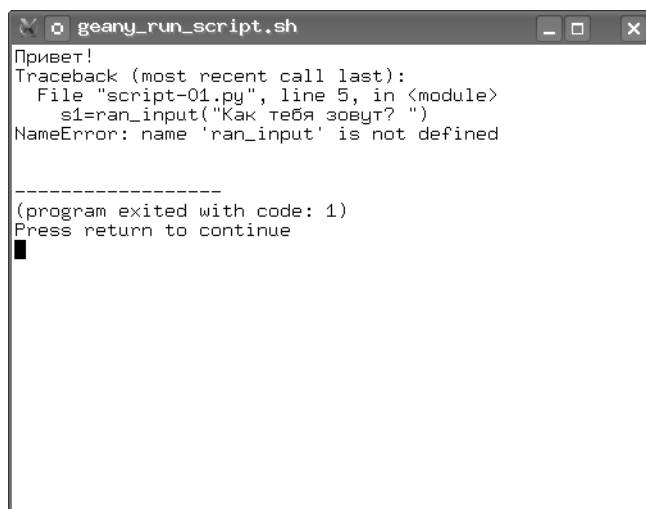
Ошибки в именах функций, ключевых словах и составных операторах сопровождаются сообщениями интерпретатора при запуске программы на выполнение.

Для каждой ошибки указывается номер строки и (по возможности) причина возникновения ошибки.



```
script-01.py X
1  # -*- coding: utf-8 -*-
2  # Простейший ввод и вывод
3  #
4  print Привет!
5  s1=raw_input("Как тебя зовут? ")
6  print "Здравствуй, "+s1+"!"
7
```

Рис. 4.21: Выделение незакрытой кавычки



```
geany_run_script.sh
Привет!
Traceback (most recent call last):
  File "script-01.py", line 5, in <module>
    s1=raw_input("Как тебя зовут? ")
NameError: name 'raw_input' is not defined

-----
(program exited with code: 1)
Press return to continue
█
```

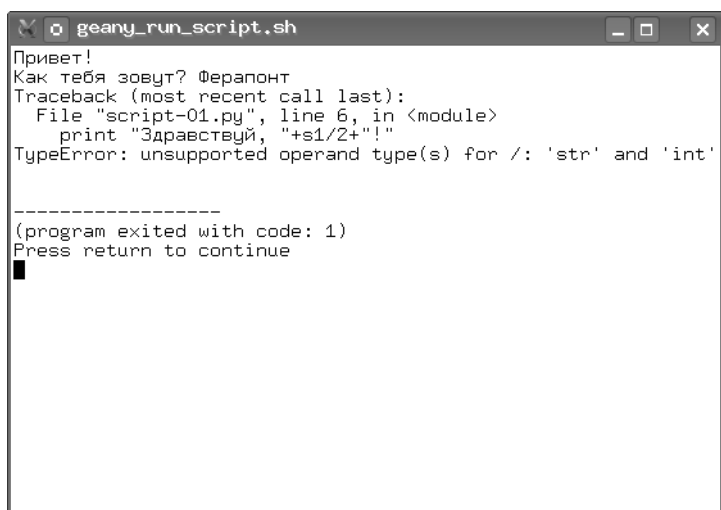
Рис. 4.22: Синтаксическая ошибка: неверное имя функции

В случае синтаксических ошибок и ошибок времени выполнения («исключения» — «exception» в терминах Python) программа завершается с кодом 1.

Если программа завершается с кодом 0 (в терминале сообщение «(program exited with code: 0)»), но работает неверно, то такая ошибка называется «семантической» и по сути является ошибкой в алгоритме. Такие ошибки выявляются только при тестировании программы на контрольных примерах.

4.5 Использование IDE Eric

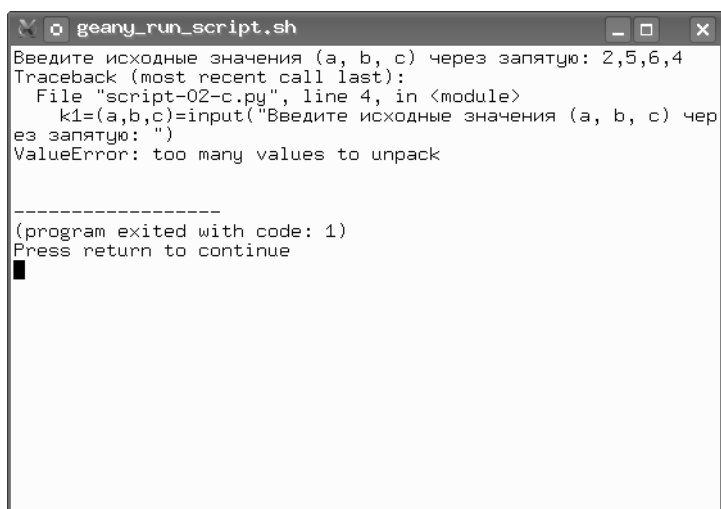
Элемент программного меню для вызова IDE Eric также находится в группе программ «Разработка». Внешний вид окна Eric при первом запуске показан на рис. 4.25. Центральная часть окна (в которой находится картинка с троллем Эриком) предназначена для размещения вкладок с текстами программ, нижняя часть — окно выполнения программы (панель «Оболочка»). В окне Eric много



```
geany_run_script.sh
Привет!
Как тебя зовут? Ферапонт
Traceback (most recent call last):
  File "script-01.py", line 6, in <module>
    print "Здравствуй, "+s1/2+"!"
TypeError: unsupported operand type(s) for /: 'str' and 'int'

-----
(program exited with code: 1)
Press return to continue
█
```

Рис. 4.23: Ошибка времени выполнения: несоответствие типов (попытка разделить строку на число)



```
geany_run_script.sh
Введите исходные значения (a, b, c) через запятую: 2,5,6,4
Traceback (most recent call last):
  File "script-02-c.py", line 4, in <module>
    k1=(a,b,c)=input("Введите исходные значения (a, b, c) чер
ез запятую: ")
ValueError: too many values to unpack

-----
(program exited with code: 1)
Press return to continue
█
```

Рис. 4.24: Ошибка времени выполнения: несоответствие количества элементов последовательности

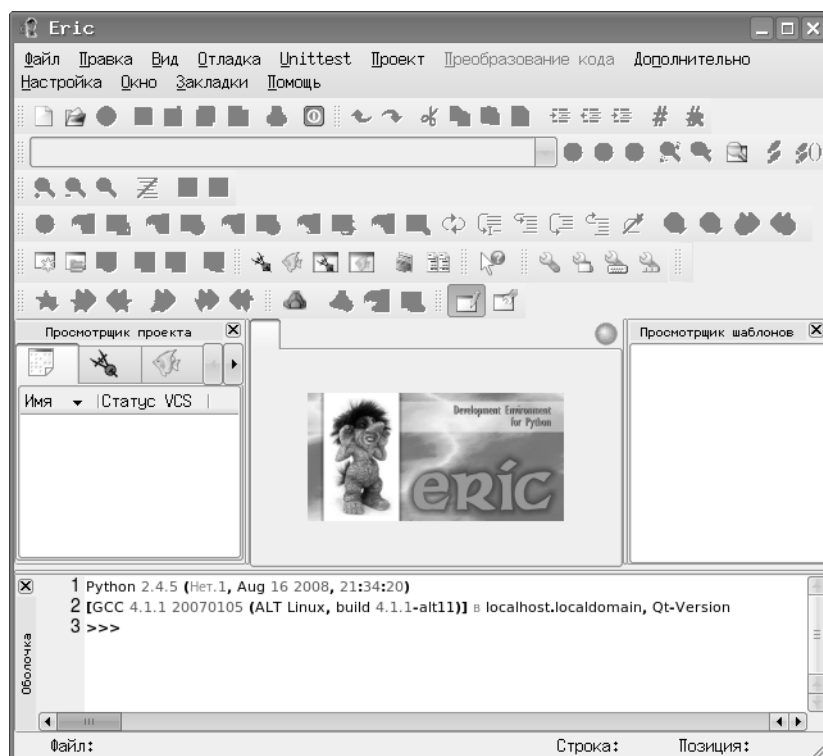


Рис. 4.25: Внешний вид IDE «Eric»

различных панелей инструментов, и очень многие кнопки недоступны (закрашены серым). Пока нет текста программы, для этих кнопок «нет работы».

4.5.1 Первоначальная настройка

Можно упростить вид окна, убрав лишние области («панели»), такие как панель «Просмотрщик проекта» слева от изображения Эрика и панель «Просмотрщик шаблонов» справа от изображения Эрика.

Нажатие кнопки с изображением пустого листа в левой части самой верхней панели инструментов (кнопка «Новый») приведёт к созданию нового документа, а внешний вид окна слегка изменится, и некоторые кнопки в панелях инструментов станут активными (рис. 4.26).

Ещё больше упростить внешний вид можно, убрав лишние панели инструментов. Если открыть пункт главного меню «Окно», то во вложенном меню «Панели инструментов» целесообразно оставить включёнными (с «галочками») панели «Закладки», «Профили», «Редактировать» и «Файл» («галочки» ставятся и уби-

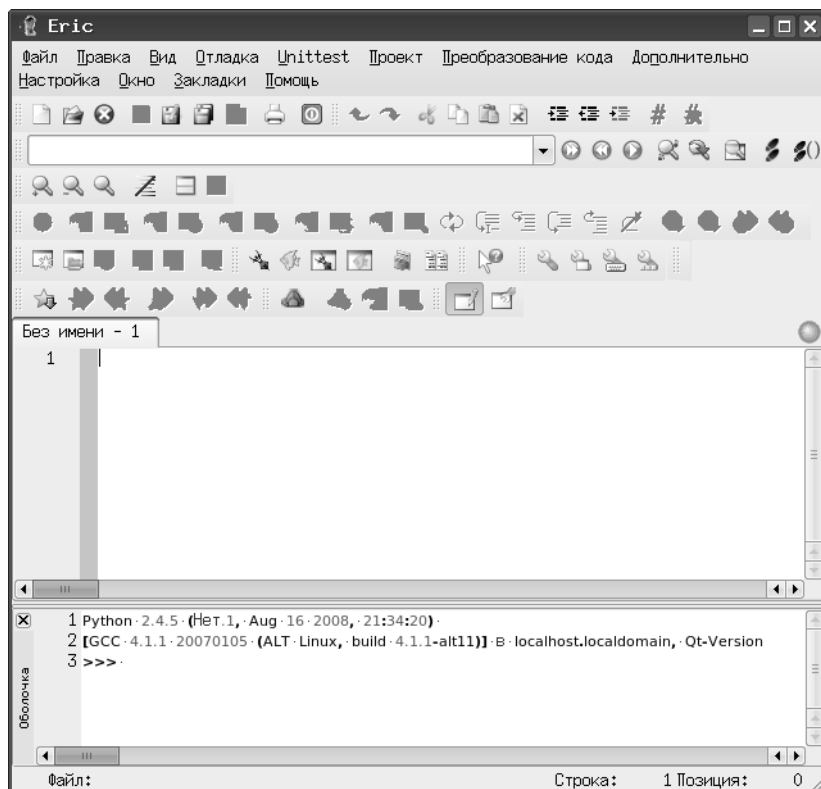


Рис. 4.26: Упрощённый Eric — редактор и оболочка

раются щелчком левой кнопкой мыши). Тогда окно IDE Eric приобретёт совсем простой вид (рис. 4.27, показан пример кода при настроенной подсветке синтаксиса).

Если хочется что-то изменить во внешнем виде программы, можно использовать настройки предпочтений («Настройка/Предпочтения...» в главном меню окна Eric). Настроек очень много (рис. 4.28), но имеет смысл пока изменять только основные настройки редактора (как показано на рис. 4.28) и стиль редактора (рис. 4.29).

В диалоге настройки основных свойств редактора полезно установить режимы показа номеров строк, полосы свёртки, пробелов и линий отступа. Полезно также включить режим подсветки скобок и автоматической проверки синтаксиса (в этом случае известные Eric слова и конструкции Python будут выделяться цветом, а неизвестные — не будут). Также полезно использование режима автоматических отступов, которые обсудим позже.

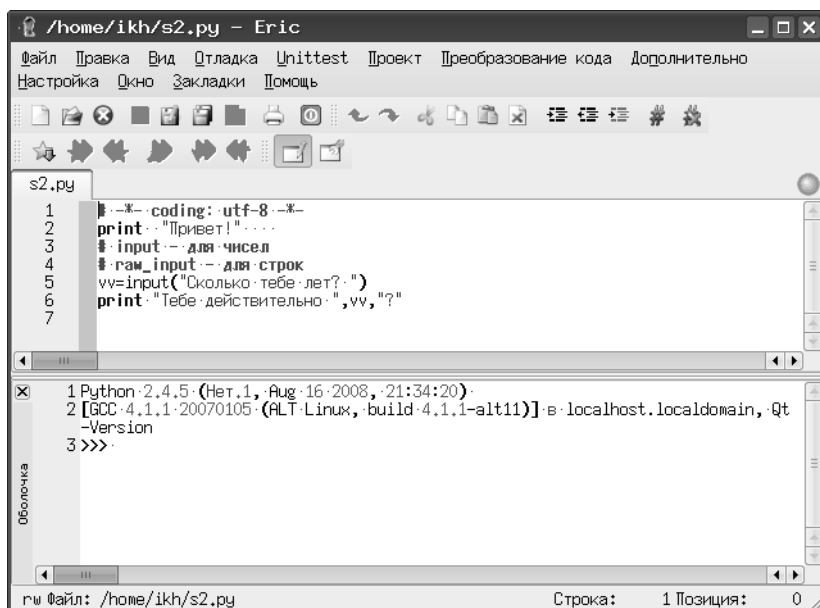


Рис. 4.27: Максимально упрощённый вид IDE Eric

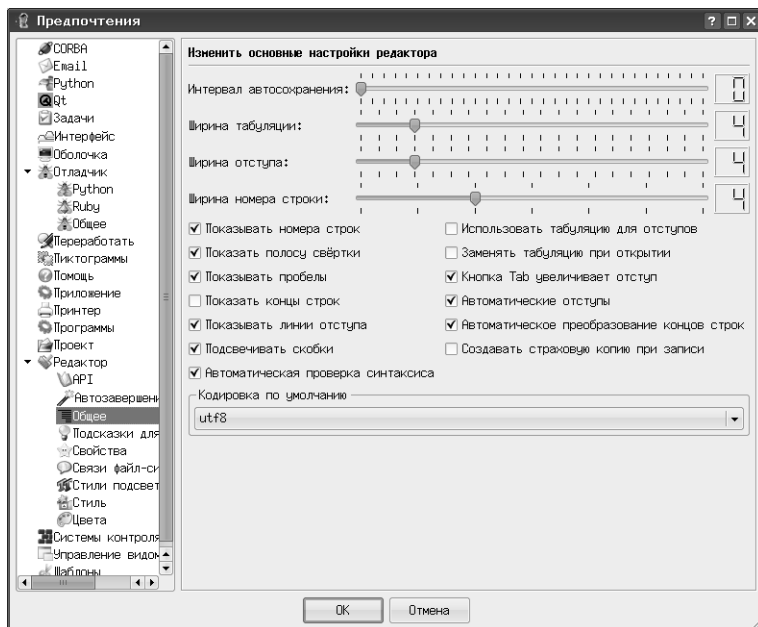


Рис. 4.28: Основные настройки редактора в IDE Eric

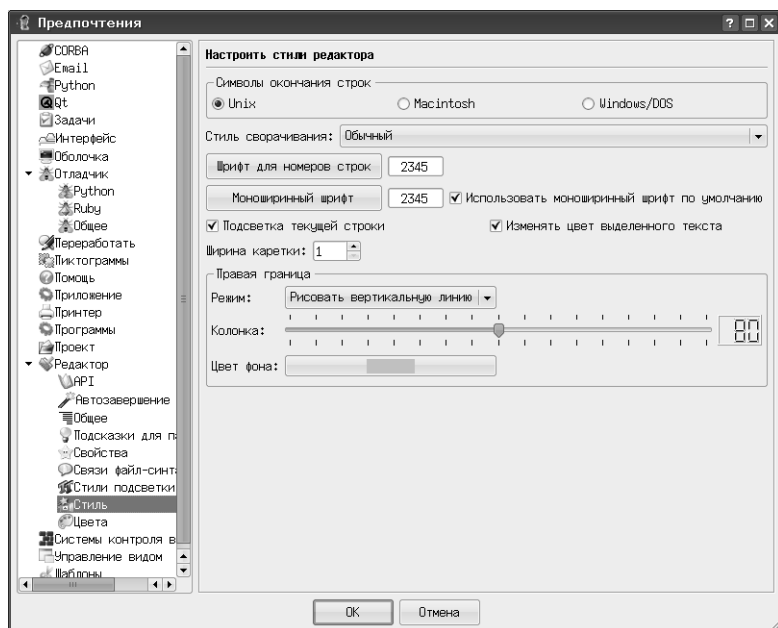


Рис. 4.29: Настройка стиля редактора в IDE Eric

В диалоге настройки стилей редактора (рис. 4.29) кнопки «Шрифт для номеров строк» и «Моноширинный шрифт» открывают диалоги выбора шрифта, в которых можно выбрать наиболее приятный для пользователя шрифт. Здесь каждый выбирает для себя, в частности, автор предпочитает для редактирования программ и для вывода сообщений IDE использовать моноширинные шрифты (в именах которых содержится слово «Fixed»).

Если есть желание изменить вид, размер и цвет подсветки для элементов языка, можно использовать диалог настройки стилей подсветки в «Настройках предпочтений», выбрав вариант «Python» из списка языков, известных лексическому анализатору (рис. 4.30).

4.5.2 Подключение документации и её использование

Очень важно правильно настроить параметры помощи и документации (диалог «Помощь», рис. 4.31). Здесь указываются варианты просмотрщика помощи IDE Eric, браузера и программы просмотра файлов PDF, а также путь к каталогу с документацией по Python. Нужно заметить, что этот каталог должен существовать, то есть документация по Python (пакет python-doc или что-то похожее) должна быть установлена в системе.

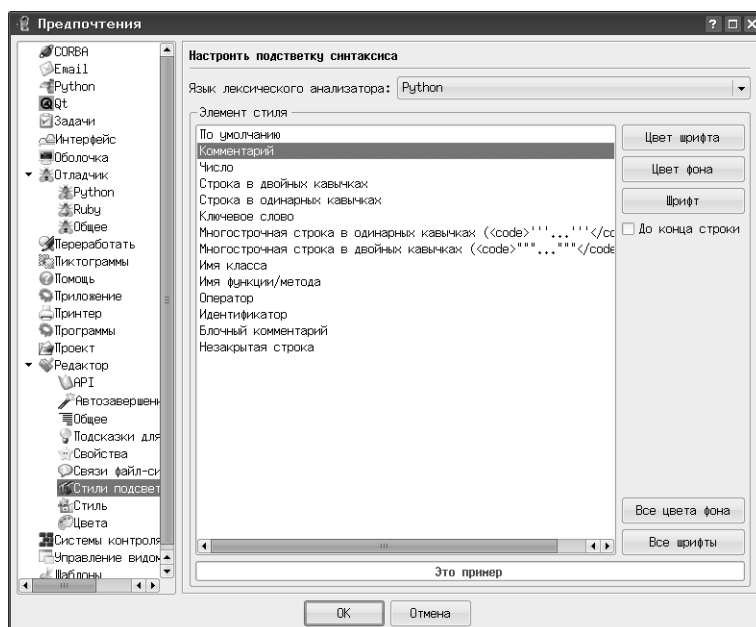


Рис. 4.30: Диалог настройки подсветки синтаксиса

Программы для просмотра web и файлов pdf можно выбрать по вкусу, узнав предварительно у администратора (или другого специалиста) как называются эти программы и где они находятся в системе.

Каталог документации по Python также нужно указывать реально существующий.

После завершения настроек попробуем воспользоваться системой помощи IDE Eric (меню «Помощь», рис. 4.32).

Выбор пункта «Просмотрщик помощи...» приводит к появлению пустого окна, в котором можно просматривать любые html-файлы. Фактически это браузер, встроенный в оболочку Eric, и его можно использовать как для работы в web, так и с файлами html в локальных каталогах.

Выбор пункта «Документация Eric» открывает документацию по пакету Eric (рис. 4.33), которая на данном этапе не нужна.

Выбор пункта «Документация Python» открывает html-руководство по Python, написанное автором языка, но, к сожалению, на английском (рис. 4.34). Это та же самая документация из пакета python-doc, которая обсуждалась ранее.

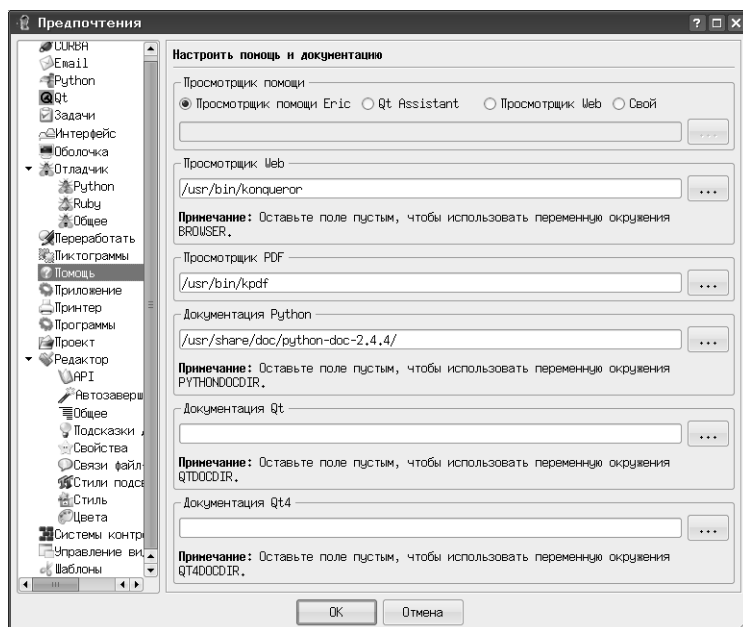


Рис. 4.31: Настройка помощи и документации

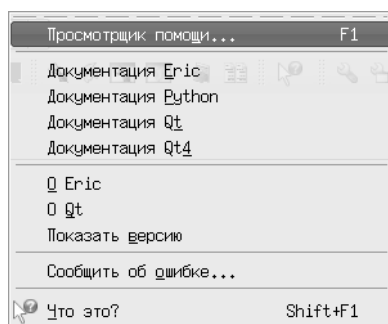


Рис. 4.32: Меню «Помощь» IDE Eric

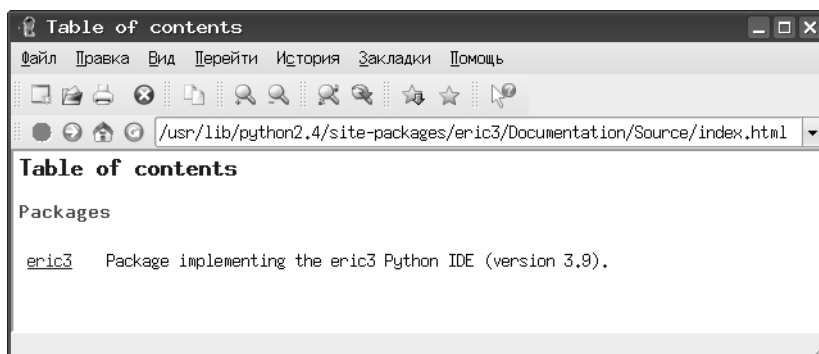


Рис. 4.33: Окно документации Eric

4.5.3 Сохранение и открытие файлов, запуск выполнения программ

Текст программы пишется в окне редактора, и пока он не сохранён, на ярлычке активной вкладки показан значок с изображением листа бумаги и карандаша (рис. 4.35).

Для сохранения файла используется команда главного меню «Файл/Сохранить» (или «Файл/Сохранить как...» при первом сохранении), что равносильно использованию комбинации клавиш **<CTRL>+S**.

Выбор этой команды открывает диалог сохранения/открытия файла (рис. 4.36). Для открытия папки (каталога) в этом диалоге нужен двойной щелчок мышью независимо от настроек пользовательской среды в сеансе.

Для открытия файла удобно использовать список последних файлов (команда «Файл/Открыть недавние файлы»).

Для запуска программы на выполнение (точнее, на трансляцию и выполнение интерпретатором Python) используется клавиша **<F2>**, нажатие на которую приводит к открытию диалога запуска программы (сценария, скрипта — рис. 4.37).

Здесь можно указать имя файла программы (программа не обязательно должна быть открыта в окне редактора) и при необходимости, аргументы скрипта. Однако для программы, открытой в активной вкладке редактора, можно не указывать имя, а сразу нажать **<ENTER>**. Таким образом, кратчайший вариант запуска выполнения программы в IDE Eric — последовательное нажатие клавиш **<F2>** и **<ENTER>**.

Если перед запуском пропущен (забыт) этап сохранения изменений — Eric напомнит и позволит сохранить последнюю версию программы и выполнить её или выполнить предыдущий вариант программы (рис. 4.38).

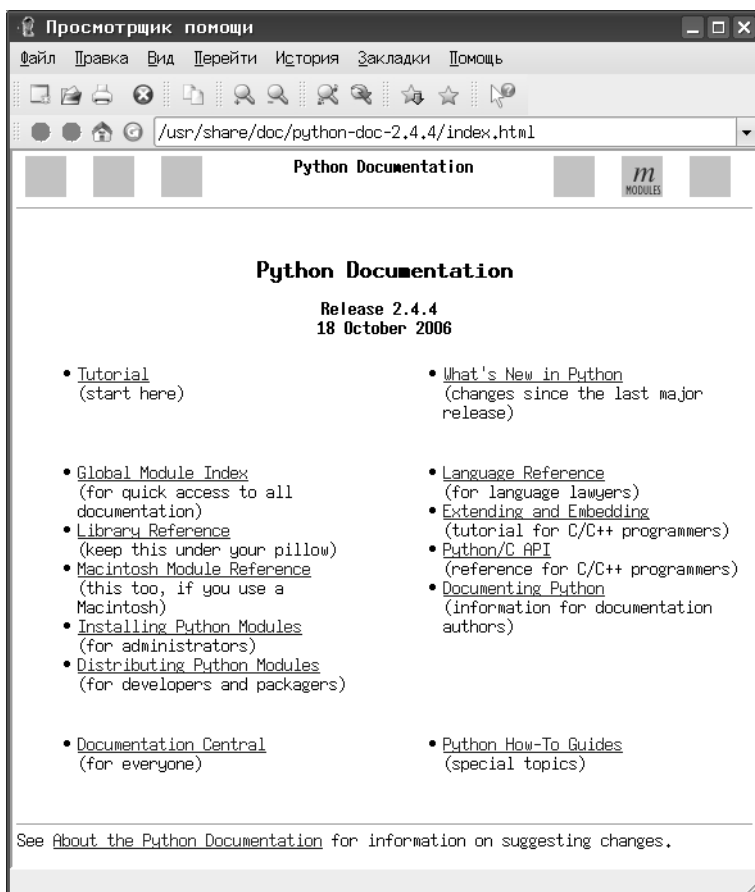


Рис. 4.34: Документация по Python в IDE Eric

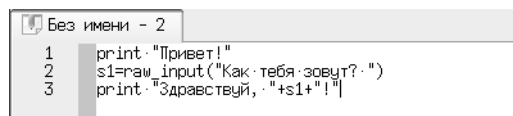


Рис. 4.35: Текст программы в окне редактора Eric до сохранения

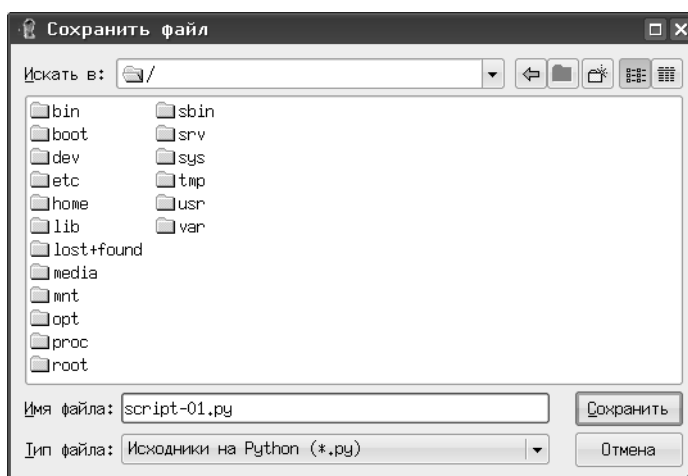


Рис. 4.36: Диалог сохранения (открытия) файла в Eric

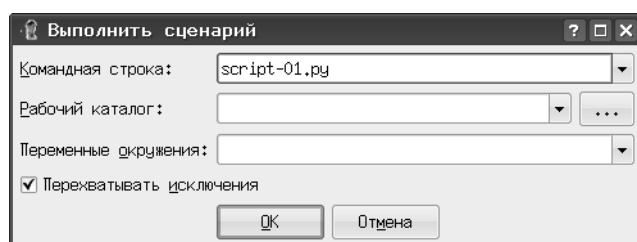


Рис. 4.37: Диалог запуска выполнения программы в Eric

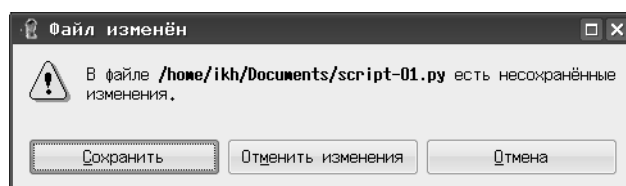


Рис. 4.38: Пример реакции IDE Eric на запуск несохранённого варианта программы

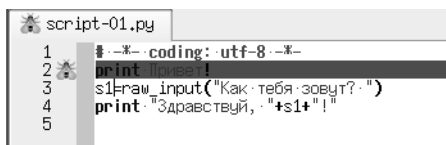


Рис. 4.39: Обработка синтаксической ошибки в IDE Eric

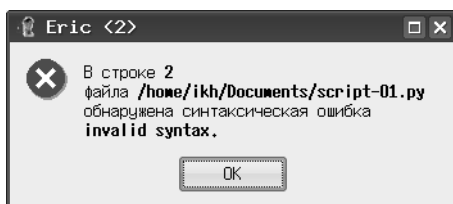


Рис. 4.40: Оповещение Eric о синтаксической ошибке

4.5.4 Обработка ошибок

Синтаксические ошибки в IDE Eric выделяются подсветкой строки с ошибкой красным цветом (или другим цветом, поведение можно настроить) и изображением «жучка» («бага») слева от номера строки (рис. 4.39). Синтаксическая ошибка определяется сразу же после перехода на новую строку в редакторе.

Если ошибка не «проявилась» в процессе написания текста программы (например, ошибка допущена при редактировании существующего текста), Eric выдаст соответствующее сообщение при попытке запуска такой программы (рис. 4.40).

Ошибки времени выполнения в Python называются «исключениями» (exception) и появляются при попытке выполнения недопустимых операций (таких как деление на 0), вводе данных с несоответствующим типом или количеством элементов, а также при попытке использования функции с неверным типом или количеством аргументов.

Такие ошибки проявляются только при выполнении программы, причём уже в процессе выполнения. IDE Eric в таких случаях также выдаёт соответствующее сообщение (рис. 4.41).

Сообщение, показанное на рис. 4.41, вызвано попыткой ввести в кортеж больше значений, чем должно в нём содержаться.

Семантические ошибки в IDE Eric (как и в любой другой среде разработки) никак не обнаруживаются и не обрабатываются, их можно обнаружить только по результатам выполнения контрольных примеров.

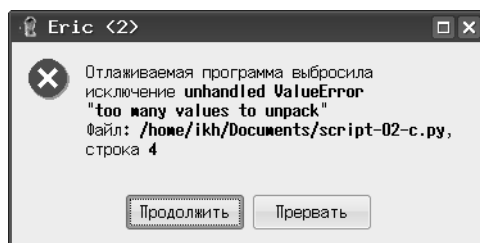


Рис. 4.41: Сообщение Eric об ошибке времени выполнения («исключении»)

4.6 Особенности работы с приложениями Tk и Tkinter

При запуске приложений Tk и Tkinter (графических) Geany открывает окно терминала и графическое окно. Для завершения работы программы нужно сначала закрывать графическое окно, а затем — окно терминала.

При работе в Eric можно получить неожиданный и неприятный эффект «зависания» при попытке закрытия приложения Tkinter кнопкой, вызывающей метод `quit` для экземпляра объекта Tk при запуске этого приложения из IDE Eric. Дело в том, что метод `quit` закрывает «родительское» окно Tk и останавливает интерпретатор Python, а при использовании IDE Eric остановить интерпретатор Python оказывается невозможно, поскольку он запущен процессом-родителем (которым является IDE Eric...). Поэтому при запуске примеров для Tkinter из IDE Eric настоятельно рекомендуется обратить внимание, что закрывать окно приложения Tkinter следует кнопкой закрытия окна пользовательской среды (крестик в правом углу в строке заголовка окна).

Ещё одна особенность касается отрисовки шрифтов в приложениях Tkinter в версиях Python до 2.5.x, которые выглядят, мягко говоря, некрасиво. Шрифт надписей для других интерфейсных элементов Tkinter можно исправить, а в области рисования (виджет `canvas`) так ничего и не удаётся сделать. Для исправления вида шрифтов интерфейсных элементов в Tkinter версий до 2.5.x можно сделать следующее.

1. Создать в «домашнем каталоге» (`/home/<username>`) с помощью любого текстового редактора (например, KWrite или Kate) файл с именем `.Xresources` (имя должно начинаться с точки!)
2. Записать в этом файле одну-единственную строку:

```
Tk*font: --terminus --r --*--12--*--*--*--*--*
```

1. Сохранить файл и перезагрузить сеанс работы (выйти из сеанса и войти снова).

Эта строка содержит название желаемого шрифта, записанное так, как оно формируется в программе xfontsel.

4.7 Использование примеров скриптов

В состав данного комплекса включены примеры программ (скриптов) на Python, обеспечивающих решение задач, описанных в «Практикуме...». Можно существенно сэкономить время на занятиях, если использовать эти примеры в качестве основы для индивидуальных заданий. Особенно это касается больших (по количеству строк) программ для работы с графикой.

Все примеры, естественно, являются свободно распространяемым программным обеспечением (если это можно так назвать) на условиях GNU GPL2 и выше.