Практикум по алгоритмизации и программированию на Python

Иван Хахаев, 2009

Часть 1. Язык Python и его особенности.

Общее описание языка Python.

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.

Функции, оформленные по определённым правилам, объединяются в модули (библиотеки функций). Модули (или некоторые функции из модулей) по мере необходимости подключаются в 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 полу-формальное 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.

Типы данных в Python.

Типы данных

Числа

Числа в Python могут быть обычными целыми (тип `int`), длинными целыми (тип `long`), вещественными (тип `float`) и комплèксными (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).

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 можно получить целое число! Для получения вещественного результата хотя бы одно из чисел должно быть вещественным. Пример: `100/8 → 12`, а вот `100/8.0 → 12.5`

`x//y`	Целочисленное деление (результат — целое число). Если оба числа в операции вещественные, получается вещественное число с дробной частью, равной нулю.
	Пример:
	`100//8 → 12`
	`101.8//12.5 → 8.0`
	(для сравнения
	`101.8/12.5 → 8.1440000000000001`)

`x%y`	Остаток от целочисленного деления `x` на `y` Пример: `10%4 → 2`

`x**y`	Возведение в степень (`x` в степени `y`). Работает и для вещественных чисел.
	Примеры:
	`2**3 → 8`
	`2.3**(-3.5) → 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 библиотеки. Все прочие функции для работы с числами (математические), такие как вычисление квадратного корня, синуса и пр. требуют подключения библиотеки `math`.

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.

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)`

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. Здесь 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 убирать 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.

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 качестве значений переменных рекомендуется использовать только символы английского алфавита. В Python строки и символы нужно заключать в кавычки (одинарные 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 резцультате 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'`).

Основные операции со строками:

Функция или операция	Описание
`len(s)`	Вычисляется длина строки `s`
`s1 + s2`	Конкатенация. К концу строки `s1` присоединяется строка `s2`, например, `«вы» + «года»` → `«выгода»`
`s * n` (или `n * s`)	`n`-кратное повторение строки `s`, например `«кан»*2` → `«канкан»`

`s[i]`	Выбор из `s` элемента с номером `i`, нумерация начинается с 0 (первый элемент имеет номер 0) Если `i<0`, отсчёт идёт с конца.
	Пример:
	`s= «дерево»`
	`s[2]` → `«р»`
	`s[-2]` → `«в»`

`s[i:j:k]`	Срез — подстрока, содержащая символы строки `s` с номерами от `i` до `j` с шагом `k` (элемент с

	номером `i` входит в итоговую подстроку, а элемент с номером `j` уже не входит). Если `k` не указан (использован вариант `s[i:j]`, то символы идут подряд (равносильно `s[i:j:1]`).
	Примеры:
	`s='derevo'`
	`s[3:5] → 'ev'`
	`s[1:5:2] → 'ee'`

`min(s)`	Определяется символ с наименьшим значением (кодом)
	Пример:
	`s="derevo"`
	`min(s) → 'd'`

`max(s)`	Определяется символ с наибольшим значением (кодом)
	Пример:
	`s="derevo"`
	`max(s) → 'v'`

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

Метод	Описание

`s1.center(n)`	Строка `s1` центрируется (дополняется пробелами слева и справа) в пространстве шириной `n` символов. Если `n < len(s1)`, пробелы не добавляются.
	Пример:
	`s1='Zoom-Zoom'`
	`s1.center(15) → '  Zoom-Zoom  '`

`s1.ljust(n)`	Строка `s1` выравнивается по левому краю (дополняется пробелами справа) в пространстве шириной `n` символов. Если `n < len(s1)`, пробелы не добавляются.
	Пример:
	`s1='Zoom-Zoom'`
	`s1.ljust(15) → 'Zoom-Zoom  '`

`s1.rjust(n)`	Строка `s1` выравнивается по правому краю (дополняется пробелами слева) в пространстве шириной `n` символов. Если `n < len(s1)`, пробелы не добавляются. Пример: `s1='Zoom-Zoom'`

	`s1.ljust(15)` → `'  Zoom-Zoom'`

`s1.count(s[,i,j])`	Определение количества вхождений подстроки `s` в строку `s1`. Можно указать позицию начала поиска `i` и окончания поиска `j` (по тем же правилам, что и начало и конец среза).
	Примеры:
	`s1='abrakadabra'`
	`s1.count('ab')` → `2`
	`s1.count('ab',1)` → `1`
	`s1.count('ab',1,-3)` → `0` потому что
	`s1[1:-3]` → `'brakada'`

`s1.find(s[,i,j])`	Поиск первого (считая слева) вхождения подстроки `s` в строку `s1`. Необязательные аргументы `i` и `j` определяют начало и конец области поиска (как в предыдущем случае).
	Пример:
	`s1='abrakadabra'`
	`s1.find('br')` → `1`

`s1.rfind(s[,i,j])`	Поиск последнего (считая слева) вхождения подстроки `s` в строку `s1`. Необязательные аргументы `i` и `j` определяют начало и конец области поиска (как в предыдущем случае).
	Пример:
	`s1='abrakadabra'`
	`s1.rfind('br')` → `8`

`s1.strip()`	Создаётся копия строки, в которой удалены пробелы в начале и в конце (если они есть или случайно образовались).
	Пример:
	`s1=' breKeKeKeKs '`
	`s2=s1.strip()`
	`s2` → `'breKeKeKeKs'`

`s1.lstrip()`	Создаётся копия строки, в которой удалены пробелы в начале (если они есть или случайно образовались).
	Пример:
	`s1=' breKeKeKeKs '`
	`s2=s1.lstrip()`
	`s2` → `'breKeKeKeKs '`

`s1.rstrip()`	Создаётся копия строки, в которой удалены пробелы в начале (если они есть или случайно образовались).
	Пример:
	`s1=' breKeKeKeKs '`
	`s2=s1.rstrip()`
	`s2` → `' breKeKeKeKs'`

`s1.replace(s2,s3[,n])`	В строке `s1` фрагмент (подстрока) `s2` заменяется на фрагмент `s3`. Необязательный аргумент `n` указывает количество замен (если требуется заменить не все фрагменты).
	Пример:
	`s1='breKeKeKeKs'`
	`ss=s1.replace('Ke','XoXo',2)`
	`ss → 'breXoXoXoXoKeKs'`

`s1.capitalize()`	Создаётся новая строка, в которой первая буква исходной строки становится заглавной (прописной), а все остальные становятся маленькими (строчными).
	Пример:
	`s1='breKeKeKeKs'`
	`s2=s1.capitalize()`
	`s2 → 'Brekekekeks'`

`s1.swapcase()`	Создаётся новая строка, в которой прописные буквы исходной строки заменяются на строчные и наоборот.
	Пример:
	`s1='breKeKeKeKs'`
	`s2=s1.swapcase()`
	`s2 → 'BREkEkEkEkS'`

`s1.upper()`	Все буквы исходной строки становятся заглавными (прописными).
	Пример:
	`s1='breKeKeKeKs'`
	`s2=s1.upper()`
	`s2 → 'BREKEKEKEKS'`

`s1.lower()`	Все буквы исходной строки становятся маленькими (строчными).
	Пример:
	`s1='breKeKeKeKs'`
	`s2=s1.lower()`
	`s2 → 'brekekekeks'`

Неизменяемые последовательности — кортежи.

Кортеж в Python — это набор разнородных элементов. Элементами кортежа могут быть числа, строки и другие структуры (в том числе и кортежи).

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 иметь совершенно одинаковые 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.

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()` в результате даёт кортеж из двух элементов.

Кортежи могут использоваться для хранения характеристик каких-то объектов. В частности, в виде кортежа можно записать фамилию ученика и его оценки за полугодие.

Основные операции с кортежами:

Функция или операция	Описание
`len(t)`	Определяется количество элементов кортежа `t`

`t1 + t2`	Объединение кортежей. Получается новый кортеж, в котором после элементов кортежа `t1` находятся элементы кортежа `t2`.
	Пример:
	`t1=(1,2,3)`
	`t2=('raz','dva')`
	`t3=t1+t2`
	`t3 → (1, 2, 3, 'raz', 'dva')`

`t * n` (или `n * t`)	`n`-кратное повторение кортежа `t`
	Пример:
	`t2=('raz','dva')`
	`t2*3 → ('raz', 'dva', 'raz', 'dva', 'raz', 'dva')`

`t[i]`	Выбор из `t` элемента с номером `i`, нумерация начинается с 0 (первый элемент имеет номер 0) Если `i<0`, отсчёт идёт с конца.
	Пример:
	`t3= (1, 2, 3, 'raz', 'dva')`
	`t3[2] → 3`
	`t3[-2] → 'raz'`

`t[i:j:k]`	Срез — кортеж, содержащий элементы кортежа `t` с номерами от `i` до `j` с шагом `k` (элемент с номером `i` входит в итоговый кортеж, а элемент с номером `j` уже не входит). Если `k` не указан (использован вариант `t[i:j]`, то элементы идут подряд (равносильно `t[i:j:1]`).

	Пример:
	`t3= (1, 2, 3, 'raz', 'dva')`
	`t3[1:4] → (2, 3, 'raz')`

`min(t)`	Определяется элемент с наименьшим значением в соответствии с алфавитным порядком.
	Пример:
	`t3= (1, 2, 3, 'raz', 'dva')`
	`min(t3) → 1`

`max(t)`	Определяется элемент с наибольшим значением в соответствии с алфавитным порядком.
	Пример:
	`t3= (1, 2, 3, 'raz', 'dva')`
	`max(t3) → 'raz'`

Важно понимать, что при определении значений минимального и максимального элементов кортежа 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.

При работе с кортежами заменить значение элемента кортежа нельзя, поэтому, если возникает такая необходимость, нужно создавать новый кортеж, используя срезы и операцию объединения.

Кортеж можно получить из строки с помощь. функции `tuple()`.

Пример:

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

Изменяемые последовательности — списки.

Список в Python — это набор разнородных элементов. Элементами списка могут быть числа, строки и другие структуры (в том числе списки и кортежи).

Самый простой способ сформировать список — перечисление его элементов в квадратных скобках через запятую, например

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

В дальнейшем при указании списков всегда будет использоваться 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 иметь совершенно

одинаковые 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, можно изменять значения элементов списка, добавлять в него элементы и удалять их.

Список может быть пустым (для его определения нужно написать `lst=[]`), а может содержать только один элемент (например, `lst=['domik']`).

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

Основные операции со списками:

Функция или операция	Описание
`len(lst)`	Определяется количество элементов списка `lst`

`lst1 + lst2`	Объединение списков. Получается новый список, в котором после элементов списка `lst1` находятся элементы списка `lst2`.
	Пример:
	`lst1=[1,2,3]`
	`lst2=['raz','dva']`
	`lst3=lst1+lst2`
	`lst3 → [1, 2, 3, 'raz', 'dva']`

`lst * n` (или `n * lst`)	`n`-кратное повторение списка `lst`
	`Пример:`
	`lst2=['raz','dva']`
	`lst2*3 → ['raz', 'dva', 'raz', 'dva', 'raz', 'dva']`

`lst[i]`	Выбор из `lst` элемента с номером `i`, нумерация начинается с 0 (первый элемент имеет номер 0) Если `i<0`, отсчёт идёт с конца.
	Пример:
	`lst3= [1, 2, 3, 'raz', 'dva']`
	`lst3[2] → 3`
	`lst3[-2] → 'raz'`

`lst[i:j:k]`	Срез — список, содержащий элементы списка `lst` с номерами от `i` до `j` с шагом `k` (элемент с номером `i` входит в итоговый список, а элемент с номером `j` уже не входит). Если `k` не указан (использован вариант `lst[i:j]`, то символы идут подряд (равносильно `lst[i:j:1]`).
	Пример:
	`lst3= [1, 2, 3, 'raz', 'dva']`
	`lst3[1:4] → [2, 3, 'raz']`

`min(lst)`	Определяется элемент с наименьшим значением в соответствии с алфавитным порядком.

	Пример:
	`lst3= [1, 2, 3, 'raz', 'dva']`
	`min(lst3) → 1`

`max(lst)`	Определяется элемент с наибольшим значением в соответствии с алфавитным порядком.
	Пример:
	`lst3= [1, 2, 3, 'raz', 'dva']`
	`max(lst3) → 'raz'`

`lst[i]=x`	Замена элемента списка с номером `i` на значение `x`. Если `x` является списком, то на место элемента списка будет вставлен список.
	Примеры:
	`lst3=[1, 2, 3, 'raz', 'dva']`
	`lst3[2]='tri'`
	`lst3 → [1, 2, 'tri', 'raz', 'dva']`
	`lst3[2]=[7,8]`
	`lst3 → [1, 2, [7, 8], 'raz', 'dva']`

`del lst[i]`	Удаление из списка элемента с номером `i`.
	Пример:
	`lst3=[1, 2, [7, 8], 'raz', 'dva']`
	`del lst3[2]`
	`lst3 → [1, 2, 'raz', 'dva']`

`lst[i:j]=x`	Замена среза списка `lst` на элемент или список `x` (несколько элементов заменяются на `x`).
	Примеры:
	`lst3=[1, 2, 3, 'raz', 'dva']`
	`lst3[2:4]='tri'`
	`lst3 → [1, 2, 't', 'r', 'i', 'dva']`
	`lst3[2:4]='a'`
	`lst3 → [1, 2, 'a', 'i', 'dva']`
	Обратите внимание, что строка воспринимается как список!

`del lst[i:j]`	Удаление элементов, входящих в указанный срез («вырезание среза»).
	Пример:
	`lst3=[1, 2, 'a', 'i', 'dva']`
	`del lst3[2:4]`
	`lst3 → [1, 2, 'dva']`

Важно понимать, что при определении значений минимального и максимального элементов списка 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.

Для списков в Python. как и для строк, определены «внутренние» методы.

Основные методы списков:

Метод	Описание

`lst.append(x)`	Добавление элемента `x` в конец списка `lst`. `x` не может быть списком. Создания нового списка не происходит.
	Пример:
	`lst=['raz','dva','tri',1,2]`
	`lst.append(3)`
	`lst → ['raz','dva','tri',1,2,3]`

`lst.extend(t)`	Добавление кортежа или списка `t` в конец списка `lst` (равносильно объединению списков, но создания нового списка не происходит).
	Пример:
	`lst1=[1,2,3]`
	`lst2=['raz','dva']`
	`lst1.extend(lst2)`
	`lst1 → [1,2,3,'raz','dva']`

`lst.count(x)`	Определение количества элементов, равных `x` в списке `lst`.
	`Пример:`
	`lst=[1, 2, 3, 'raz', 'dva', 'raz', 'dva']`
	`lst.count('raz') → 2`

`lst.index(x)`	Определение первой слева позиции элемента `x` в списке `lst`. Если такого элемента нет, возникает сообщение об ошибке.
	Пример:
	`lst=[1, 2, 3, 'raz', 'dva', 'raz', 'dva']`
	`lst.index('dva') → 4`

`lst.remove(x)`	Удаление элемента `x` в списке `lst` в первой слева позиции. Если такого элемента нет, возникает сообщение об ошибке.
	Пример:
	`lst=[1, 2, 3, 'raz', 'dva', 'raz', 'dva']`
	`lst.remove('dva')`
	`lst → [1,2,3,'raz','raz','dva']`

`lst.pop(i)`	Удаление элемента с номером `i` из списка `lst`. При этом выдаётся значение этого элемента («извлечение» элемента из списка). Если номер не

	указан, удаляется последний элемент.
	Примеры:
	`lst=[1,2,3,'raz','raz','dva']`
	`lst.pop(3) → 'raz'`
	`lst → [1,2,3,'raz','dva']`
	`lst.pop() → 'dva'`
	`lst → [1,2,3,'raz']`

`lst.insert(i,x)`	Вставка элемента или списка `x` в позицию `i` списка `lst`. Если `i>=0`, вставка идёт в начало списка. Если `i>len(lst)`, вставка идёт в конец списка. Новый список не создаётся.
	Пример:
	`lst= [1,2,3,'raz']`
	`lst.insert(3,'tri')`
	`lst → [1,2,3,'tri','raz']`

`lst.sort()`	Сортировка списка по возрастанию (в алфавитном порядке). Новый список не создаётся.
	Пример:
	`lst=[1,2,3,'tri','raz']`
	`lst.sort()`
	`lst → [1,2,3,'raz','tri']`

`lst.reverse()`	Замена порядка следования элементов на обратный. Новый список не создаётся.
	Пример:
	`lst=[1,2,3,'raz','tri']`
	`lst.reverse()`
	`lst → ['tri','raz',3,2,1]`

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.

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()`.

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,t4)
t → [0.20000000000000001, 0.33333333333333331, 0.42857142857142855]
```

В случае, если в функции `map()` в качестве первого аргумента используется специальная функция `None`, результат равносилен 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.

Есть ещё две важные функции взаимного преобразования строк и списков (вообще говоря, они являются методами строк).

Функция (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']
```

Функция (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'
```

Специальный список range().

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 `[x0,x1)` 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. Часть 2. Глава «Циклические алгоритмы. Обработка последовательностей и одномерных массивов»).

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

Примеры:

```
sum(range(10)) → 45

sum(range(0,15,3)) → 30
```

Ввод и вывод в Python

Диалоговый режим

При работе с интерактивными оболочками или в процессе организации взаимодействия программы с пользователем («диалога») для ввода чисел и, соответственно, определения значений переменных используется 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. При использовании русских слов в строке-подсказке нужно указывать кодировочную таблицу (см. главу «Программа на Python»). Строка-подсказка может быть в двойных или в одиночных кавычках. Для выполнения операций `input()` или `raw_input()` интерпретатор останавливает программу и после строки-подсказки требуется ввести число и нажать <ENTER>. Если строка подсказки отсутствует, будет показан курсор оболочки в пустой строке окна выполнения. Если требуется ввести несколько чисел, из нужно вводить через запятую и нажимать <ENTER> только после последнего введённого числа.

В примерах значок *>>>* используется в качестве приглашения интерактивной оболочки. наклонный шрифт показывает то, что следует набирать на клавиатуре, а символ # означает комментарий (пояснение).

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

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 значение результатом функции `input()` является 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')
>>>


>>> 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)
>>>
```

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 так называемый «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')
```

Прочитаем строку из файла:

```
>>> s=fd.read()
>>> s
'1 2 3\n'
>>>
```

Теперь 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.

Повторение операции чтения данных:

```
>>> s=fd.read()
>>> s
'a b c\n'
>>>
```

Повторное чтение данных из этого же файла выдало новую строку. Таким образом, каждая операция чтения даёт следующую строку файла.

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

close():

```
>>> fd.close()
```

Теперь 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')
>>> s2='c d e'
>>> fd.write(s2)
>>> fd.close()
```

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. Если это не сделать, в файл ничего не запишется.

Теперь рассмотрим случай создания файла и добавления данных в него.

```
>>> fd=open('2.dat','a')
>>> s1='c d e\n'
>>> fd.write(s1)
>>> s2='3 4 5\n'
>>> fd.write(s2)
>>> fd.close()
```

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
```

Среда программирования Python.

Python имеет возможность работы в режиме интерпретатора, в котором команды и операции выполнятся 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 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>.

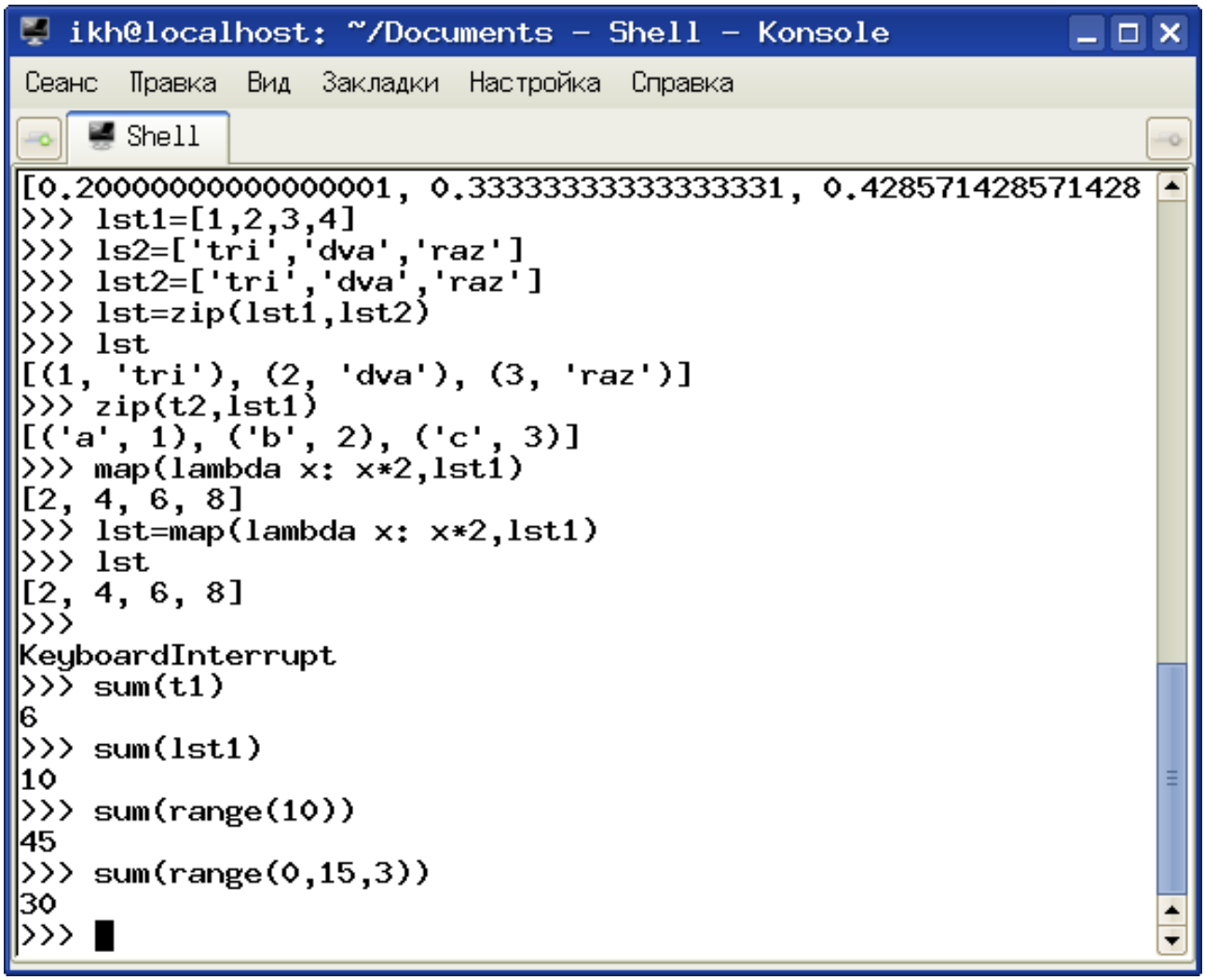

Рисунок 1. Сеанс в интерпретаторе Python

Показанные выше примеры как раз и являлись фрагментами сеансов работы в интерактивной оболочке.

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 достоиств 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. 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).

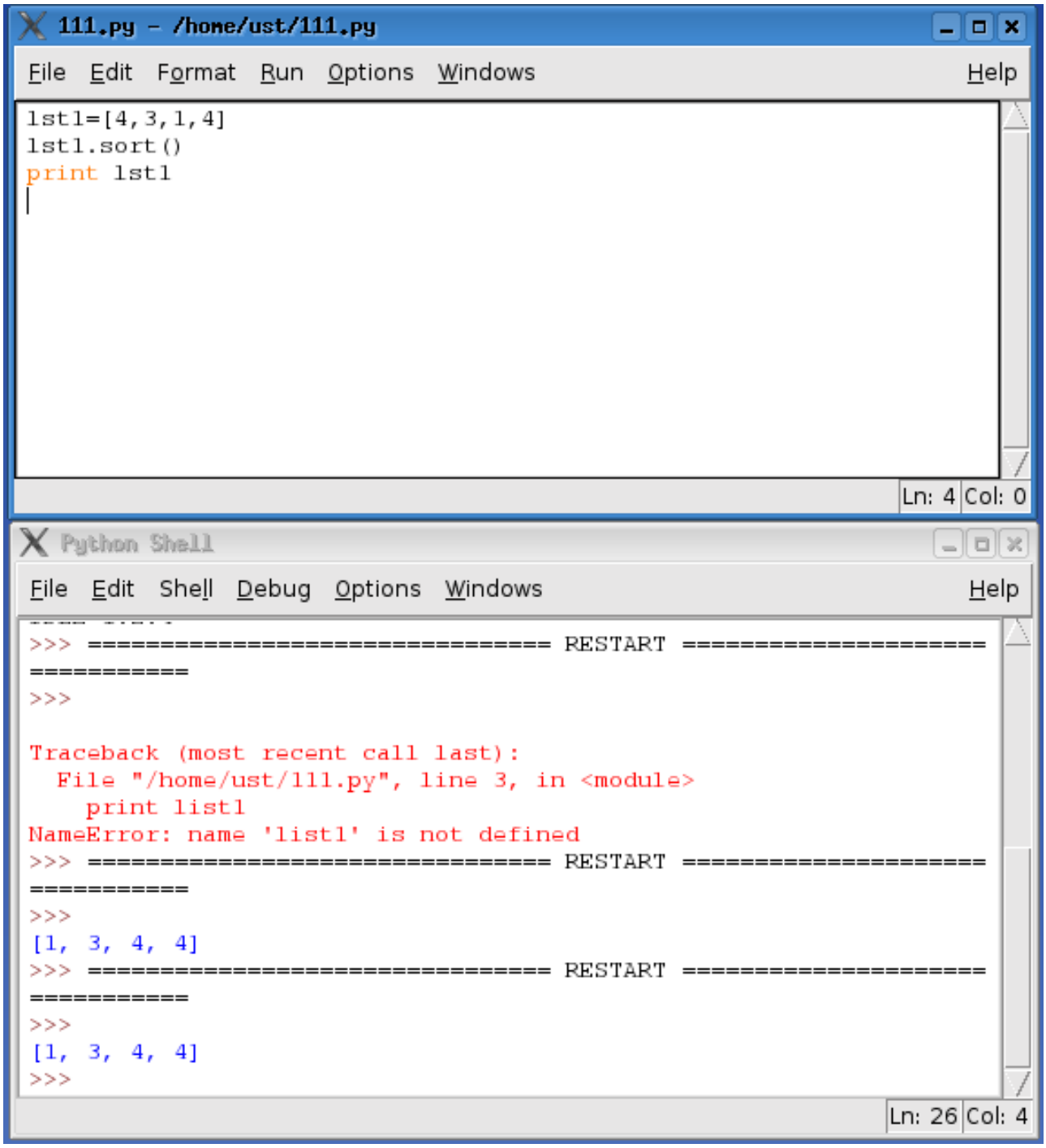

Рисунок 2. Интегрированная среда разработки IDLE

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. 3), которая обладает большим количеством настроек и возможностей отладки программ и больших программных проектов.

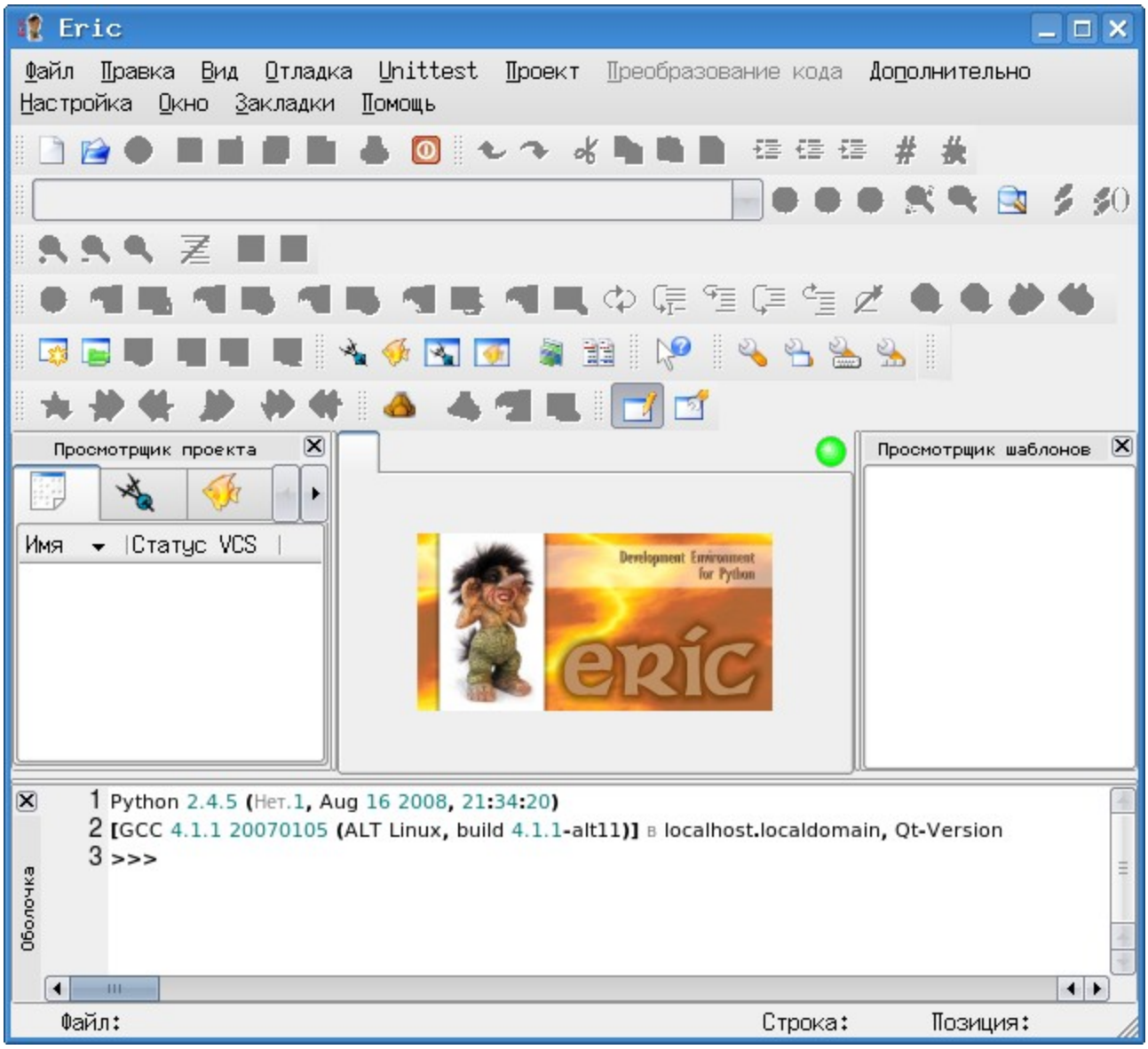

Рисунок 3. IDE Eric

Однако эту среду разработки (точнее, её внешний вид) можно существенно упростить и сделать её более понятной для начинающих (рис. 4

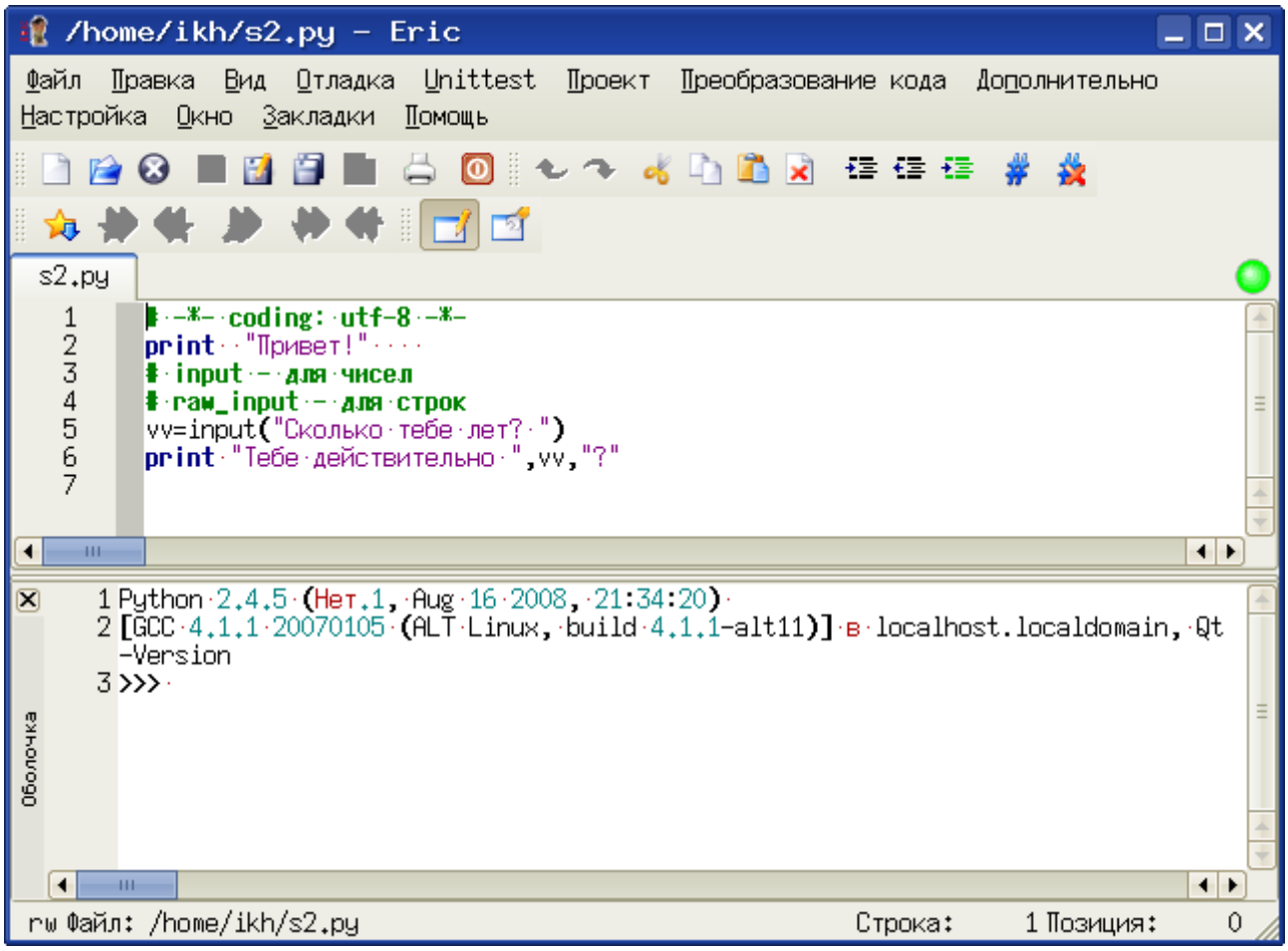

Рисунок 4. Упрощённый вариант IDE Eric

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

Программа на Python

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 поиск ошибок написания команд. Файл с программой должен иметь «расширение» `.py` (например, `my_program.py`).

Первую строку программы полезно оформить как комментарий, в котором указывается кодировочная таблица для символов кириллицы.

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 имеет собственный вариант выполнения программы. В IDLE для этого нужно в окне редактора нажать клавишу <F5>, а в Eric — клавиши <F2> и <ENTER>.

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 английским особенных 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 Pyhton, 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>

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`.

IDE Eric с помощью команд главного меню «Помощь/Документация Python» позволяет получить справку по Python в виде гипертекста (рис. 5) и тоже на английском языке.

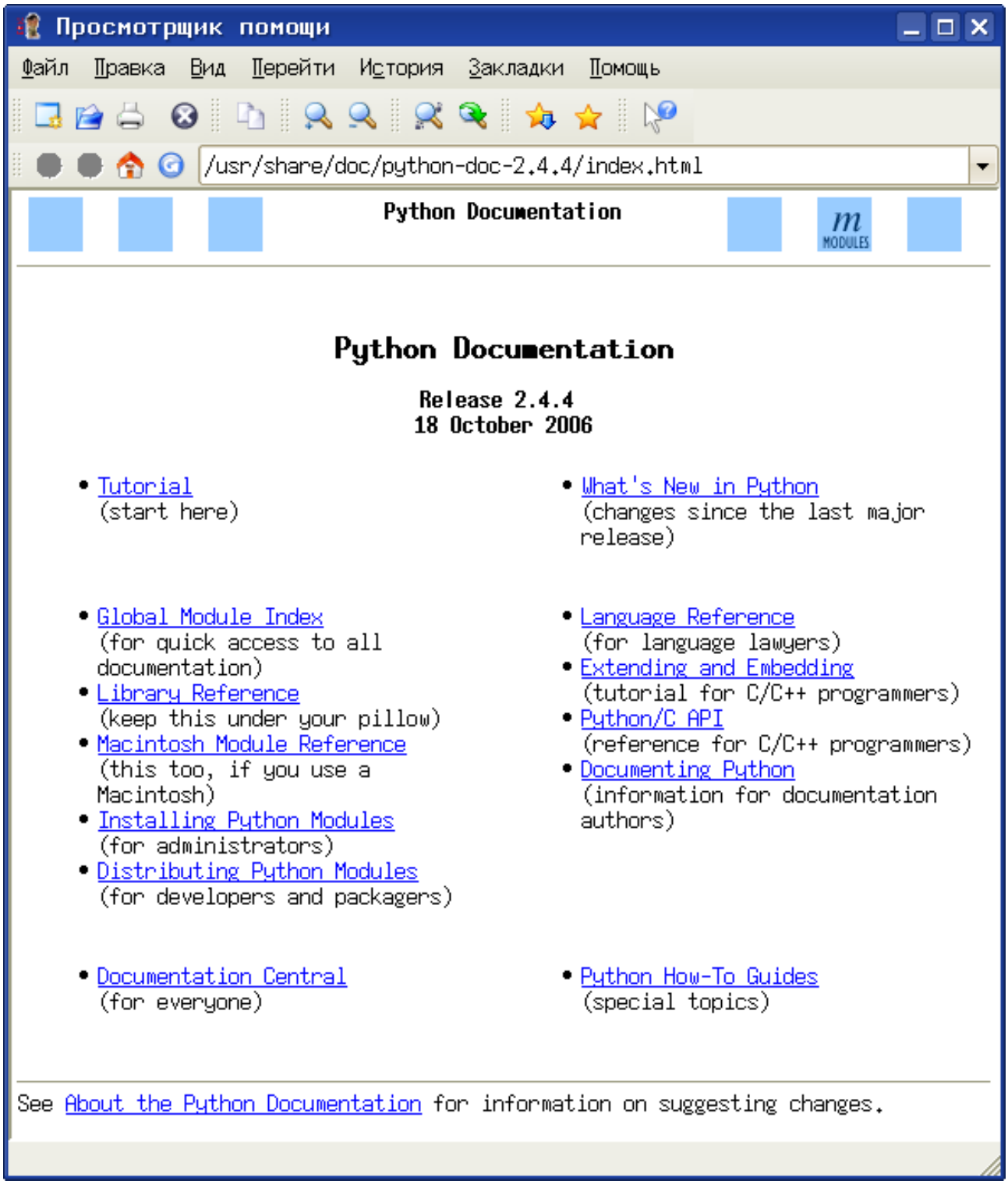

Рисунок 5. Справка по Python в IDE Eric

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

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 ограничение на длину строки установлено 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 два способа изменить порядок элементов кортежа из четырёх элементов на противоположный.