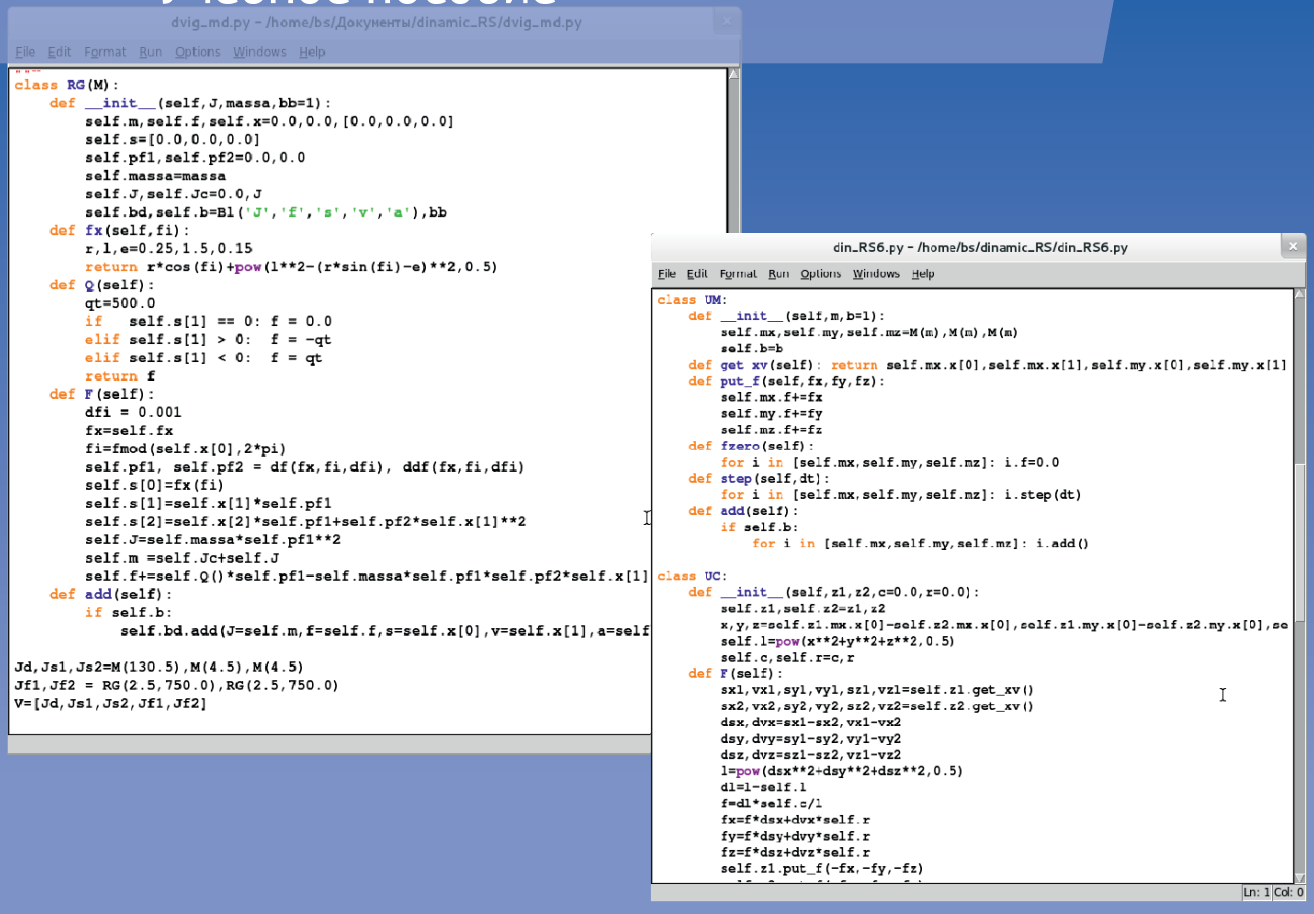
Уральский федеральный университет
имени первого Президента России Б.Н.Ельцина
Механико-машиностроительный институт

С. К. БУЙНАЧЕВ
Н. Ю. БОКЛАГ

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ PYTHON

Учебное пособие



Министерство образования и науки Российской Федерации

Уральский федеральный университет
имени первого Президента России Б. Н. Ельцина

С. К. Буйначев, Н. Ю. Боклаг

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ PYTHON

Рекомендовано методическим советом УрФУ
*в качестве **учебного пособия** для студентов,*
обучающихся по направлениям подготовки
151000 «Технологические машины и оборудование»,
190100 «Наземные транспортно-технологические комплексы»,
190600 «Эксплуатация транспортно-технологических машин
и комплексов»

Екатеринбург
Издательство Уральского университета
2014

УДК 004.432Python(075.8)
ББК 32.973.26-018.1Pythonя73
Б90

Рецензенты: доц., д-р техн. наук Е. Е. Баженов (Уральский государственный экономический университет);

доц., канд. техн. наук В. П. Подогов (Российский государственный профессионально-педагогический университет)

Научный редактор – доц., канд. техн. наук Ю. В. Песин

Буйначев, С. К.

Б90 Основы программирования на языке Python : учебное пособие / С. К. Буйначев, Н. Ю. Боклаг. – Екатеринбург : Изд-во Урал. ун-та, 2014. – 91, [1] с.

ISBN 978-5-7996-1198-9

Пособие содержит начальные сведения о программировании на языке Python и является основой для изучения курса «Численные методы и оптимизация». Собраны сведения из книг таких известных авторов, как Г. Россум, М. Лутц, Р. Сузи, Д. Бизли, А. Лесса. Предложен новый подход к использованию баз данных для накопления результатов расчета с дальнейшим анализом и визуализацией решений.

Может быть рекомендовано студентам различных специальностей технических вузов, занимающихся программированием, математическим моделированием и численными методами, а также может служить справочным материалом при выполнении курсовых и дипломных работ, связанных с расчетами на компьютере.

Библиогр.: 10 назв.

УДК 004.432Python(075.8)
ББК 32.973.26-018.1Pythonя73

ISBN 978-5-7996-1198-9

ОГЛАВЛЕНИЕ

1. ЗАПУСК ОБОЛОЧКИ ПРОГРАММ И ИНСТРУКЦИИ ЯЗЫКА PYTHON

Программы Python выполняются интерпретатором. На компьютерах с системами Unix и Linux интерпретатор можно вызвать, набрав команду **python.** В системах Windows и Macintosh интерпретатор можно запустить как приложение (либо из меню Start, либо двойным щелчком на пиктограмме интерпретатора). После запуска интерпретатора появляется подсказка, в которой можно начать отладку операторов программы в простом цикле чтения/выполнения. Например, в приведенном ниже выводе интерпретатор отображает сообщение об авторских правах и предоставляет пользователю подсказку >>>, в которой пользователь набирает знакомую команду "Hello World":

```
Python 1.5.2 (#0, Jun I 1999, 20:22:04)
Copyright 1991-1995 Stichting Mathematisch Centrum, Amsterdam
>>> print "Hello World"
Hello World
>>>
```

Программы можно также помещать в файл:

```
# helloworld.py
print "Hello World"
```

Исходные файлы Python имеют расширение *.py. Символ # в предыдущем примере обозначает комментарий, который продолжается до конца строки.

В системе Windows программы Python можно запускать двойным щелчком на файле с расширением .py. При этом происходит запуск интерпретатора и выполнение программы в окне терминала. В таком случае окно терминала немедленно исчезает после того, как программа завершает свое выполнение (чаще всего прежде, чем удается прочитать ее вывод). Чтобы избежать этого, можно воспользоваться средой интегрированной разработки (Idle или Pythonwin). Альтернативным методом является запуск программы с использованием пакетного файла с расширением *.bat, python -i helloworld.py, содержащего оператор типа python -i helloworld.py, который указывает интерпретатору, чтобы он перешел в интерактивный режим после

выполнения программы. Можно также изменить расширение файла на *.pyw, что в Windows означает запуск как исполняемого файла (без использования консоли).

В системе Macintosh программы можно выполнять из встроенной среды интегрированной разработки. Кроме того, утилита BuildApplet (включенная в дистрибутив) позволяет преобразовать программу Python в документ, который автоматически запускается интерпретатором при его открытии.

В интерпретаторе программу можно выполнить с помощью функции execfile(), как показано в следующем примере:

```
>>> execfile("helloworld.py")
Hello World
```

В системе Unix можно также вызывать Python с использованием символов **#!** в сценарии командного интерпретатора:

```
#!/usr/local/bin/python
print "Hello World"
```

Интерпретатор продолжает работу до тех пор, пока не достигнет конца входного файла. При интерактивном выполнении можно выйти из него, введя символ EOF (end of file – конец файла) или выбрав Exit из выпадающего меню (если оно имеется). В Unix в качестве символа EOF служит <Ctrl+D>; в Windows – <Ctrl+Z>.

Из программы можно также выйти, вызвав функцию sys.exit() или активизировав исключение **SystemExit** (это эквивалентно). Например:

```
>>> import sys
>>> sys.exit()
```

или

```
>>> raise SystemExit
```

В программе Python выделяются следующие ступени иерархии:

- программы делятся на модули;
- модули содержат инструкции;
- инструкции состоят из выражений;
- выражения создают и обрабатывают объекты.

Инструкции в языке Python приведены ниже.

Инструкция	Роль	Пример
1	2	3
Присваивание	Создание ссылок	a,b,c='ножницы','бумага', 'камень'
Вызовы	Запуск функций	f.write('Пролог\n')
print	Вывод на консоль	print 'Знание – сила'
if/elif/else	Операция выбора	if 'python' in text: print text
for/else	Обработка последовательности в цикле	for x in thelist: print x
while/else	Цикл общего назначения	while x>y: y+=1
pass	Пустая инструкция	if a: pass
break, continue	Переходы в теле цикла	while 1: if not in line: break
try/except/ finally	Обработка исключений	try: action() except: print 'action error'
raise	Возбуждение исключений	raise endSearch, location
import, from	Доступ к модулям	import sys from sys import stdin

def, return, yield	Создание функции	def f(a,b,c=1,*d):
		return a+b+c+d[0]
		def gen(n):
		for i in n,
		yield i*2

class	Описание класса	class subclass(Superclass): staticData=[]
global	Пространство имен	def function(): global x,y x='new'

Окончание таблицы

1	2	3
del	Удаление ссылок	del data[k] del data[i:j] del obj.attr del variable
exec	Запуск фрагментов программного кода	exec 'import '+modName exec code in gdict, ldict
assert	Отладочные проверки	assert x > y
with/as	Менеджеры контекста	with open('data') as myfile: process(myfile)

2. ВСТРОЕННЫЕ ТИПЫ ДАННЫХ

Python – это язык с динамическим контролем типа, в котором имена во время выполнения программы могут представлять значения различных типов. И действительно, имена, используемые в программе, – это только метки для различных величин и объектов. Оператор присваивания просто создает связь между именем и значением. В этом состоит одно из отличий данного языка, например, от C, в котором имена представлены объектами с постоянным размером и размещением в памяти, где находятся результаты.

Все данные в Python представлены объектами. Имена являются лишь ссылками на эти объекты и не несут нагрузки по декларации типа. Значения встроенных типов имеют специальную поддержку в синтаксисе языка: можно записать литерал строки, числа, списка, кортежа, словаря (и их разновидностей). Синтаксическую же поддержку операций над встроенными типами можно легко сделать доступной и для объектов определяемых пользователями классов.

Следует также отметить, что объекты могут быть *неизменчивыми* и *изменчивыми.* Например, строки в Python являются неизменчивыми, поэтому операции над строками создают новые строки.

Карта встроенных типов (с именами функций для приведения к нужному типу и именами классов для наследования от этих типов):

1. Специальные типы: None, NotImplemented и Ellipsis;
2. Числа:

1) целые:
   - обычное целое int;
   - целое произвольной точности long;
   - логическое bool;
2) число с плавающей точкой float;
3) комплексное число complex.

3. Последовательности:
   1) неизменчивые:
      - строка str;
      - Unicode-строка Unicod;
      - кортеж tuple;
   2) изменчивые:
      - список list;
      - отображения;
      - словарь dict.
4. Объекты, которые можно вызвать:
   1) функции (пользовательские и встроенные);
   2) функции-генераторы;
   3) методы (пользовательские и встроенные);
   4) классы (новые и «классические»);
   5) экземпляры классов (если имеют метод _call_).
5. Модули.
6. Файлы file.
7. Вспомогательные типы buffer, slice.

Узнать тип любого объекта можно с помощью встроенной функции type().

2.1. Тип int и long

Два типа: int (целые числа) и long (целые произвольной точности) – служат моделью для представления целых чисел. Первый соответствует типу long в компиляторе C для используемой архитектуры.

Числовые литералы можно записать в системах счисления с основанием 8, 10 или 16:

```
# В этих литералах записано число 10
print 10, 012, 0xA. 10L
```

Набор операций над числами достаточно стандартный как по семантике, так и по обозначениям:

```
>>> print 1 + 1, 3 – 2, 2*2, 7/4, 5%3
>>> print 2L ** 1000
```

Значения типа int должны покрывать диапазон от 2147483648 до 2147483647, а точность целых произвольной точности зависит от объема доступной памяти.

2.2. Тип float

Соответствует С-типу double для используемой архитектуры. Записывается вполне традиционным способом либо через точку, либо в нотации с экспонентой:

```
>>>pi=3.1415926535897931
>>>pi**40 7.691214220515699e+19
```

2.3. Тип bool

Подтип целочисленного типа для канонического обозначения логических величин. Два значения: True(истина) и False(ложь) – вот и все, что принадлежит этому типу. Любой непустой и ненулевой объект Python имеет значение True.

2.4. Тип string и тип Unicode

В Python строки бывают двух типов: обычные и Unicode-строки. Фактически строка – это последовательность символов (в случае обычных строк можно сказать «последовательность байтов»). Строки-константы можно задать в программе с помощью строковых литералов. Для литералов наравне используются как апострофы «'», так и обычные двойные кавычки «"». Для многострочных литералов можно использовать утроенные апострофы или утроенные кавычки. Управ-

ляющие последовательности внутри строковых литералов задаются обратной косой чертой (\). Примеры написания строковых литералов:

```
s1 = "строка 1"
s2 = 'строка2\n с переводом строки внутри'
s3 = """строка3
с переводом строки внутри"""
ul = u'\u043f\u0440\u0438\u0432\u0435\u' # привет
u2 = u'Еще пример'    # не забудьте про coding!
```

Для строк имеется еще одна их разновидность – *необработанные* строковые литералы. В этих литералах обратная косая черта и следующие за ней символы не интерпретируются как спецсимволы, а вставляются в строку как есть:

```
my_re = r"(\d)=\1"
```

Набор операций над строками включает конкатенацию **«+»**, повтор «*», форматирование «%». Также строки имеют большое количество методов, некоторые из них приведены ниже. Полный набор методов (и их необязательных аргументов) можно получить в документации по Python.

```
>>> "A" + "B"
'AB'
>>>"A"*10
'AAAAAAAAAA'
>>> "%s %i" % ("abc", 12)
'abc 12'
```

Некоторые методы строковых объектов будут рассмотрены далее.

2.5. Тип tuple

Для представления константной последовательности (разнородных) объектов используется тип кортеж. Литерал кортежа обычно записывается в круглых скобках, но можно, если не возникают неоднозначности, писать и без них. Примеры записи кортежей:

```
p= (1.2, 3.4, 0.9)
p1 = 1, 3, 9   # без скобок
p2 = 3, 8, 5,  # запятая в конце игнорируется
```

Использовать синтаксис кортежей можно и в левой части оператора присваивания. В этом случае на основе вычисленных справа значений формируется кортеж и связывается один в один с именами в левой части. Поэтому обмен значениями записывается очень компактно:

```
c, a, b = a, b, c
```

2.6. Тип list

В «чистом» Python нет массивов с произвольным типом элемента. Вместо них используются списки. Их можно задать с помощью литералов, записываемых в квадратных скобках, или посредством списковых включений. Варианты задания списка приведены ниже:

```
lst1 = [1, 2, 3,]
lst2 = [x**2 for x in range<10) if x % 2 == 1]
lst3 = list("abcde")
```

Для работы со списками существует несколько методов, дополнительных к тем, что имеют неизменчивые последовательности. Все они связаны с изменением списка.

2.7. Последовательности

Ниже обобщены основные методы последовательностей. Следует напомнить, что последовательности бывают неизменчивыми и изменчивыми. У последних методов чуть больше.

Синтаксис	Семантика
len(s)	Длина последовательности s
x in s	Проверка принадлежности элемента последовательности. В новых версиях Python можно проверять принадлежность подстроки строке. Возвращает True или False

Окончание таблицы

x not in s	== not x in s
s + sl	Конкатенация последовательностей
s*n или n*s	Последовательность из n раз повторенной s. Если n < 0, то возвращается пустая последовательность
s[i]	Возвращает i-й элемент s или len(s) + i-й, если i < 0
s[i:j:d]	Срез из последовательности s от i до j с шагом d
min(s)	Наименьший элемент s
max(s)	Наибольший элемент s
s[i]=x	i-й элемент списка s заменяется на x
s[i:j:d]= t	Срез от i до j (с шагом d) заменяется на (список) t
del s[i:j:d]	Удаление элементов среза из последовательности

Методы изменчивых последовательностей

Метод	Описание
append(x)	Добавляет элемент в конец последовательности
count(x)	Считает количество элементов, равных x
extend(s)	Добавляет к концу последовательности последовательность s
index(x)	Возвращает наименьшее i, такое что s[i]==x. Возбуждает исключение ValueError, если x не найден в s
insert(i, x)	Вставляет элемент x в i-й промежуток
pop([i])	Возвращает i-й элемент, удаляя его из последовательности
reverse()	Меняет порядок элементов s на обратный
sort([cmpfunc])	Сортирует элементы s. Может быть указана своя функция сравнения cmpfunc

Взятие элемента по индексу и срезы

Здесь же следует сказать несколько слов об индексировании последовательностей и выделении подстрок (и вообще – подпоследовательностей) по индексам. Для получения отдельного элемента последовательности используются квадратные скобки, в которых стоит выражение, дающее индекс. Индексы последовательностей в Python начинаются с нуля. Отрицательные индексы служат для отсчета элементов с конца последовательности (-1 – последний элемент). Пример:

```
>>> s = [0, 1, 2, 3, 4]
>>> print s[0], s[-l], s[3]
043
>>> s[2] = -2
>>> print s
[0, 1, -2, 3, 4]
>>> del s[2]
>>> print s
[0, 1, 3, 4]
```

Примечание. Удалять элементы можно только из изменчивых последовательностей и категорически не делать этого внутри цикла при их обработке.

Есть особенности при работе со срезами. В Python при взятии среза последовательности принято нумеровать не элементы, а промежутки между ними. Поначалу это кажется необычным, тем не менее очень удобно для указания произвольных срезов. Перед нулевым (по индексу) элементом последовательности промежуток имеет номер 0, после него будет 1 и т. д. Отрицательные значения отсчитывают промежутки с конца строки. Для записи срезов используется следующий синтаксис:

последовательность[нач:кон:шаг]

где нач – промежуток начала среза; кон – конца среза; шаг – шаг. По умолчанию нач = 0, кон = 1en (последовательность); шаг = 1. Если шаг не указан, второе двоеточие можно опустить. Приведем пример работы со срезами:

```
>>> s = range(10)
>>> s
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> s[0:3]
[0, 1, 2]
>>> s[-1:]
[9]
>>> s[::3]
[0, 3, 6, 9]
>>> s[0:0] = [-1, -1, -1]
>>> s
[-1, -1, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> del s[:3]
>>> s
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Как видно из этого примера, с помощью срезов удобно задавать любую подстроку, даже если она нулевой длины, как для удаления элементов, так и для вставки в строго определенное место.

2.8. Тип dict

Словарь (хэш, ассоциативный массив) – это изменчивая структура данных для хранения пар ключ-значение, где значение однозначно определяется ключом. В качестве ключа может выступать неизменчивый тип данных (число, строка, кортеж и т. п.). Порядок пар ключ-значение произволен. Ниже приведен литерал для словаря и пример работы со словарем:

```
d = {1: 'one', 2: 'two', 3: 'three', 4: 'four'}
d0 = {0: 'zero'}
print d[l]              # берется значение по ключу
d[0] =0                 # присваивается значение по ключу
del d[0]                # удаляется пара ключ-значение с данным ключом
print d
for key, val in d.items():        # цикл по всему словарю
  print key, val for key in d.keys () :          # цикл по ключам словаря
```

```
    print key, d[key] for val in d.values ():          # цикл по значениям словаря
    print val
d.update(d0)#пополняется словарь из другого      print len(d)
```

2.9. Тип file

Объекты этого типа предназначены для работы с внешними данными. В простом случае – это файл на диске. Файловые объекты должны поддерживать основные методы: read(), write(), readline(), readlines(), seek(), tell(), close() и т. п. Копирование файла:

```
f1=open("file1.txt","r")
f2=open("file2.txt","w")
for line in f1.readlines(): f2.write(line)
f2.close()
f1.close()
```

Кроме собственно файлов, в Python используются и файлоподобные объекты. В очень многих функциях просто неважно, передан ли ей объект типа file или другого типа, если он имеет все те же методы (и в том же смысле). Например, можно достигнуть копирования содержимого по ссылке (URL) в файл file.txt, если заменить первую строку на

```
import urllib
f1 = urilib.urlopen("http://python.onego.ru")
```

3. ВЫРАЖЕНИЯ

В современных языках программирования принято производить большую часть обработки данных в выражениях. Синтаксис выражений у многих языков программирования примерно одинаков. Синтаксис выражений Python не удивит программиста чем-то новым. (Разве что цепочечные сравнения могут приятно порадовать.)

Приоритет операций показан в таблице (в порядке уменьшения). Для унарных операций x обозначает операнд. Ассоциативность операций в Python – слева направо.

Операция	Название
lambda	Лямбда-выражение
or	Логическое ИЛИ
and	Логическое И
not x	Логическое НЕ
in, not in	Проверка принадлежности
is, is not	Проверка идентичности
<, <=, >, >=, !=, ==	Сравнения
|	Побитовое ИЛИ
^	Побитовое исключающее ИЛИ
&	Побитовое И
<<, >>	Побитовые сдвиги
+, -	Сложение и вычитание
*, /, %	Умножение, деление, остаток
+x, -x	Определение и смена знака
~x	Побитовое НЕ
**	Возведение в степень
x.атрибут	Ссылка на атрибут
x[индекс]	Взятие элемента по индексу
x[от:до]	Выделение среза (от и до)
f(аргумент)	Вызов функции
(...)	Скобки или кортеж
[...]	Список или списковое включение
{кл:зн,...}	Словарь пар ключ-значение
'выражения'	Преобразование к строке (repr)

Порядок вычислений операндов определяется правилами:

1. Операнд слева вычисляется раньше операнда справа во всех бинарных операциях, кроме возведения в степень.

2. Цепочка сравнений вида a < b < c < … < y < z фактически равносильна данному выражению:

(a < b) and (b < c) and ... and (y < z).

3. Перед фактическим выполнением операции вычисляются нужные для нее операнды. В большинстве бинарных операций предварительно вычисляются оба операнда (сначала левый), но операции or и and, а также цепочки сравнений вычисляют такое количество операндов, которого достаточно для получения результата. В невычисленной части выражения в таком случае могут даже быть неопределенные имена. Это важно учитывать, если используются функции с побочными эффектами.

4. Аргументы функций, выражения для списков, кортежей, словарей и т. п. вычисляются слева направо, в порядке следования в выражении. В случае неясности приоритетов желательно применять скобки, несмотря на то что одни и те же символы могут использоваться для разных операций, приоритеты которых не меняются. Так, «%» имеет тот же приоритет, что и «*», а потому в следующем примере скобки просто необходимы, чтобы операция умножения произошла перед операцией форматирования:

```
print "%i" % (i*j)
```

Имена используются так, как если бы они были определены в текущем модуле:

```
os.system("dir")
digits = re.compile("\d+")
print argv[0], environ
```

Повторный импорт модуля происходит гораздо быстрее, так как модули кэшируются интерпретатором. Загруженный модуль можно загрузить еще раз (например, если файл с текстом модуля изменился на диске) с помощью функции reload():

```
import mymodule reload(mymodule)
```

Однако в этом случае все объекты, являющиеся экземплярами классов из старого варианта модуля, не изменят своего поведения.

4. ФУНКЦИИ

Для создания функции применяется оператор def, как показано в следующем примере:

```
def remainder(a,b):
        q = a/b
        r = a – q*b
        return r
print (3.0,2.0)
```

Для того чтобы вызвать функцию, нужно просто указать имя функции, за которым следуют ее параметры, заключенные в круглые скобки**.** Для возврата из функции нескольких значений может применяться кортеж, как показано ниже:

```
def divide(a,b):
        q = a/b #Если a и b – целые,q – целое.
        r = a – q*b
        return (q,r)
a,b=divide(3.0,2.0)
print a,b
```

Для того чтобы присвоить параметру значение, принятое по умолчанию, можно использовать оператор присваивания:

```
def connect(hostname,port,timeout=300):
        # Тело функции
```

Если в определении функции даны значения, принятые по умолчанию, их можно опускать в последующих вызовах функции. Например:

```
connect('www.python.org', 80)
```

Можно также вызывать функции, используя ключевые параметры и указывая параметры в произвольном порядке. Например:

```
connect(port=80,hostname="www.python.org")
```

При создании или присваивании значений переменных внутри функции область их определения является локальной. Для изменения значения глобальной переменной внутри функции используется оператор global следующим образом:

```
a = 4.5
def foo () :
        global aa = 8.8
        # Изменяет глобальную переменную a
```

5. ВСТРОЕННЫЕ ФУНКЦИИ

В среде Python без дополнительных операций импорта доступно более сотни встроенных объектов, в основном функций и исключений. Для удобства функции можно условно разделить на следующие категории:

Категория	Функции
1	2
Функции преобразования типов и классы	coerce, str, repr, int, list,tuple,long,float, complex, dict, super, file, bool, object
Числовые и строковые функции	abs, divmod, ord, pow, len, chr, unichr,hex,oct,cmp,round, Unicode
Функции обработки данных	apply, map, filter, reduce, zip, range, xrange, max, min, iter, enumerate, sum
Функции определения свойств	hash, id, callable, issubclass, isinstance, type
Функции для доступа к внутренним структурам	locals, globals, vars, intern, dir

Окончание таблицы

1	2
Функции компиляции и исполнения	eval, execfile, reload, __import__, compile
Функции ввода-вывода	input, raw_input, open
Функции для работы с атрибутами	getattr, setattr, delattr, hasattr
Функции-«украшатели» методов классов	staticmethod, classmethod, property
Прочие функции	buffer, slice

Уточнить назначение функции, ее аргументов и результата можно в интерактивной сессии интерпретатора Python:

```
>>> help(len)
Help on built-in function len:
len(...)
len(object) -> integer
Return the number of items of a sequence or mapping.
## Или так:
>>> print len.__doc__
len(object) -> integer
Return the number of items of a sequence or mapping.
```

6. КЛАССЫ

Оператор class применяется для определения объектов новых типов и для объектно-ориентированного программирования. Например, следующий класс определяет простой стек:

```
class Stack:
        def _init_(self):     # Инициализировать стек
                self.stack = [ ]
        def push(self,object):
                self.stack.append(object)
        def pop(self, object):
```

```
            return self.stack.pop()
        def length(self):
            return ten(self.stack)
```

В классе метод определяется с помощью оператора def. Первый параметр каждого метода всегда ссылается на сам объект. В соответствии с общепринятым соглашением для этого параметра применяется имя self. Все операции, затрагивающие атрибуты объекта, должны явно ссылаться на переменную self. Методы с ведущими и конечными двойными подчеркиваниями являются специальными методами. Метод (конструктор) __init__ применяется для инициализации объекта при его создании.

Для того чтобы использовать класс в целях создания объекта, можно применить код:

```
s = Stack()    # Создать стек
s.push("Dave") # Продвинуть в него некоторые объекты
s.push(42)
s.push([3,4,5])
x = s.pop()    # x получает значение [3,4,5]
y = s.pop()    # y получает значение 42
del s          # Уничтожить s
```

7. ИСКЛЮЧЕНИЯ

Если в программе возникает ошибка, то активизируется исключение и появляется сообщение об ошибке, как в следующем примере:

```
Traceback (Innermost last):
File "<interactive input>", line 42, in foo.py NameError: a
```

В сообщении об ошибке указан тип возникшей ошибки, а также место ее возникновения. Обычно ошибки приводят к аварийному завершению программы. Однако можно перехватывать и обрабатывать исключения с помощью операторов try и except:

```
try:
        f = open("file.txt","r")
```

```
except IOError, e:
        print e
```

При возникновении ошибки IOError подробные сведения о причинах ошибки помещаются в переменную **e** и управление передается коду в блоке except. При возникновении исключения какого-то другого вида оно передается во включающий блок кода (если он есть). Если ошибки не возникают, код в блоке except игнорируется.

Для сообщения об исключении используется оператор raise. Для активизации исключения можно воспользоваться одним из встроенных исключений следующим образом:

```
raise RuntimeError, "Unrecoverable error"
```

8. ФУНКЦИИ ПРЕОБРАЗОВАНИЯ ТИПОВ И КЛАССЫ

Функции и классы из этой категории служат для преобразования типов данных. В старых версиях Python для преобразования к нужному типу использовалась одноименная функция. В новых версиях Python роль таких функций играют имена встроенных классов (однако семантика не изменилась). Пример:

```
>>> int(23.5)
23
>>> float('12.345')
12.345000000000001
>>> dict([('a', 2), ('b', 3)])
{'a': 2, 'b': 3}
>>> object
<type 'object'>
```

9. ЧИСЛОВЫЕ И СТРОКОВЫЕ ФУНКЦИИ

Функции работают с числовыми или строковыми аргументами. Ниже даны описания этих функций.

Функции	Описание
abs(x)	Модуль числа x
divmod(x, y)	Частное и остаток от деления
pow(x, y[,m])	Возведение x в степень y
round(n[, z])	Округление чисел до заданного знака после точки
ord(s)	Функция возвращает код заданного символа в строке
chr(n)	Возвращает строку с символом с заданным кодом
len(s)	Возвращает число элементов последовательности
oct(n), hex(n)	Возвращают строку с восьмеричным или шестнадцатеричным представлением целого числа n
cmp(x, y)	Сравнение двух значений. Результат: отрицательный, ноль или положительный, в зависимости от результата сравнения
Unicode(s[, encoding[, errors]])	Создает Unicode-объект, соответствующий строке s в заданной кодировке encoding

Ошибки кодирования обрабатываются в соответствии с errors, который может принимать следующие значения:

'strict' – строгое преобразование;

'replace' – с заменой несуществующих символов;

'ignore' – игнорировать несуществующие символы).

По умолчанию: encoding=' utf-8' , errors='strict'.

Следующий пример строит таблицу кодировки кириллических букв в Unicode:

```
print "Таблица Unicode (русские буквы)". center(18*4) i = 0
for c in"АБВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ"\
        "абвгдежзийклмнопрстуфхцчшщъыьэюя":
        u = Unicode(c, 'koi8-r')
        print "%3i: %1s %s"  %  (ord(u) , c, 'u.'),
        i += 1
        if i % 4 == 0: print
```

10. ФУНКЦИИ ОБРАБОТКИ ДАННЫХ

В программе Python предопределены функции генерации массива чисел.

Пример с функциями range() и enumerate():

```
>>> for i,c in enumerate("ABC"):  print i, c
0 A
1 B
2 C
>>> print range(4, 20, 2)
[4, 6, 8, 10, 12, 14, .16, 18]
```

11. ФУНКЦИИ ОПРЕДЕЛЕНИЯ СВОЙСТВ

Функции определения свойств обеспечивают доступ к некоторым встроенным атрибутам объектов и другим свойствам. Следующий пример показывает некоторые из этих функций:

```
>>> s = "abcde"
>>> s1 ="abcde"
>>> s2 = "ab" + "cde"
>>> print "hash:", hash(s), hash(sl), hash(s2)
hash: -1332677140 -1332677140 -1332677140
>>> print "id:", id(s), id(sl), id(s2)
id: 1076618592 1076618592 1076618656
```

Здесь можно увидеть, что для одного и того же строкового литерала "abcde" получается один и тот же объект, тогда как для одинаковых по значению объектов вполне можно получить разные объекты.

12. ФУНКЦИИ ДЛЯ ДОСТУПА К ВНУТРЕННИМ СТРУКТУРАМ

В используемой реализации языка Python глобальные и локальные переменные доступны в виде словаря благодаря функциям globals() и locals(). Изменять эти словари не рекомендуется.

Функция vars() возвращает таблицу локальных имен некоторого объекта (если параметр не задан, она возвращает то же, что и locals()):

```
a=1
b=2
c=3
print "%(a)s + %(b)s = %(c)s" % vars()
```

13. ФУНКЦИИ КОМПИЛЯЦИИ И ИСПОЛНЕНИЯ

Функция reload() уже рассматривалась, а из остальных функций этой категории особого внимания заслуживает eval(). Эта функция вычисляет переданное ей выражение. Пример:

```
a=2
b=3
for op in "+-*/%":
        e = "a" + op + "b"
        print e, "->", eval(e)
```

У функции eval(), кроме подлежащего вычислению выражения, есть еще два параметра – с их помощью можно задать глобальное и локальное пространства имен, из которых будут разрешаться имена выражения. Ниже представлен предыдущий пример для использования с собственным словарем имен в качестве глобального пространства имен:

```
for op in "+-*/%":
        e = "a" + op + "b"
        print e, "->", eval(e, {'a': 2, 'b': 3})
```

В целях безопасности не следует применять eval() для аргумента, в котором присутствует непроверенный ввод от пользователя.

14. ФУНКЦИИ ВВОДА-ВЫВОДА

Функции input() и raw_input() используются для ввода со стандартного ввода. Функция open() служит для открытия файла по имени для чтения, записи или изменения.

Функция принимает три аргумента. Первые два – имя файла (путь к файлу), режим открытия ("r" – чтение, "w" – запись, "a" – добавление или "w+", "a+", "r+" – изменение; также может прибавляться «t», что обозначает текстовый файл и имеет значение только на платформе Windows). Третий аргумент указывает режим буферизации: 0 без буферизации; 1 построчная буферизация; больше 1 буфер указанного размера в байтах.

15. ВВОД И ВЫВОД ФАЙЛОВ

В следующей программе выполняется открытие файла и построчное чтение его содержимого:

```
f = open("data.txt") #Возвращает объект-файл
line = f.readline()  #Вызывает метод readline() файла
while line:
        print line
        line = f.readline()
f.close ()
```

Функция open() открывает новый объект файла. Вызывая методы этого объекта, можно выполнять различные операции с файлом. Метод readline() считывает одну строку ввода, включая завершающий символ новой строки. В конце файла возвращается пустая строка. Аналогичным образом можно использовать метод write() для записи результатов программы вычисления сложного процента в файл:

```
f = open("out","w")      # Открыть файл для записи
while year <= numyears:
        principal = principal*(1+rate)
        f.write("%3d %0.2f\n" % (year,principal))
        year = year + 1
f.close()
```

Функция open() возвращает объект файла, который поддерживает методы, перечисленные ниже.

Метод	Описание
f.read([n])	Читать не более n байтов
f.readline()	Читать одну строку ввода
f.readlines()	Читать все строки и возвратить *список*
f.write(S)	Писать строку S
f.writelines(L)	Писать все строки в списке L
f. close()	Закрыть файл
f.tell()	Возвратить текущий указатель файла
f.seek(offset [, where])	Установить указатель на новую позицию в файле
f.isatty()	Возвратить 1,если f – диалоговый терминал
f.flush()	Вывести данные на устройство и очистить буфер вывода
f.truncate([size])	Усечь файл до размера в байтах не более size
f.fileno()	Возвратить целочисленный дескриптор файла

16. СТАНДАРТНЫЕ ФАЙЛЫ ВВОДА/ВЫВОДА ДАННЫХ И ВЫВОДА ОШИБОК

В интерпретаторе предусмотрены три стандартных объекта файла, которые известны под названием стандартного файла ввода данных, стандартного файла вывода данных и стандартного файла вывода ошибок и доступны в модуле sys под именами sys.stdin, sys.stdout и sys.stderr соответственно. В частности, stdin – это объект файла, соответствующий входному потоку символов, поступающему в интерпретатор, stdout – это объект файла, который получает вывод, выработанный оператором print, а stderr – это файл, который получает сообщения об ошибках. Чаще всего stdin предполагает ввод с клавиатуры, а stdout и stderr – вывод текста на экран.

Методы, описанные в предыдущей главе, могут применяться для выполнения ввода/вывода произвольных данных с участием

пользователя. Например, следующая функция считывает строку ввода из стандартного файла ввода:

```
def gets():
    text = " "
      while 1:
            c = sys.stdin.read(1)
            text = text + c
            if c == '\n' :
            break
      return text
```

Для чтения строки текста из файла stdin может также применяться встроенная функция raw_input(prompt):

```
s = raw_input("type something : ")
print "You typed '%s'" % (s,)
```

И наконец, прерывания с клавиатуры (для выработки которых часто применяется комбинация клавиш <Ctrl+C>) активизируют исключение KeyboardInterrupt, которое может быть перехвачено с использованием обработчика исключений.

17. ФУНКЦИИ ДЛЯ РАБОТЫ С АТРИБУТАМИ

У объектов в языке Python могут быть атрибуты (в терминологии языка C++ – члены-данные и члены-функции). Следующие две программы эквивалентны:

```
# первая программа:
class A: pass
a = A()
a.attr = 1
try: print a.attr
except: print None
del a.attr
# вторая программа:
class A: pass
```

```
a = A()
setattr(a, 'attr', 1)
if hasattrfa, 'attr'): getattr(a, 'attr')
else: print None
delattr(a, 'attr')
```

18. МОДУЛИ

По мере того как программы возрастают в размерах, может возникнуть необходимость разбить их на несколько файлов для удобства сопровождения. Для этого в языке Python предусмотрена возможность поместить в файл определения и использовать их в качестве модуля, который можно импортировать в другие программы и сценарии. Для создания модуля нужно поместить соответствующие операторы и определения в файл, имеющий такое же имя, как у модуля.

Примечание. Файл должен иметь расширение .py.

Например:

```
# файл :   div.py
def divide(a,b):
   q = a/b #Если a и b – целые, то q – целое число
   r = a – q*b
   return (q,r)
```

Для того чтобы использовать этот модуль в других программах, можно воспользоваться оператором import:

```
import div
a, b = div.divide(2305, 29)
```

Оператор import создает новое пространство имен, которое содержит все объекты, определенные в модуле. Для доступа к этому пространству имен можно просто использовать имя модуля в качестве префикса, как в случае div.divide() из предыдущего примера. Для выполнения импорта конкретных определений в текущее пространство имен служит оператор from:

```
from div import divide
a,b = divide(2305,29) # Префикс div больше не нужен
```

Для загрузки всего содержимого модуля в текущее пространство имен можно также использовать такую конструкцию:

```
from div import *
```

Функция dir() выводит на экран содержимое модуля и является полезным средством экспериментирования в интерактивном режиме:

```
>>> import string
>>> dir(string)
['__builtins__', '__doc__', '__file__', '__name__', '_idmap', '_idmapL',
'_lower', '_swapcase', '_upper', 'atof, 'atof_error', 'atoi', 'atoi_error', 'atol',
'atol_error', 'capitalize', 'capwords', 'center', 'count', 'digits', 'expandtabs',
'find',
```

19. МОДУЛИ СТАНДАРТНОЙ БИБЛИОТЕКИ

Модули стандартной библиотеки условно разбиты на группы по тематике.

Группа	Модули (пакеты)
Сервисы периода выполнения	sys, atexit, copy, traceback, math, cmath, random, time, calendar, datetime, sets, array, struct, itertools, locale, gettext
Поддержка цикла разработки	pdb, hotshot, profile, unittest, pydoc (docutils, distutils)
Взаимодействие с ОС	os, os.path, getopt, glob, popen2, shutil, select, signal, stat, tempfile
Обработка текстов	string, re, StringIO, codecs, difflib, mmap, sgmllib, htmllib, htmlentitydefs (xml)
Многопоточные вычисления	threading, thread, Queue
Хранение данных. Архивация	pickle, shelve, anydbm, gdbm, gzip, zlib, zipfile, bz2, csv, tarfile

Окончание таблицы

Группа	Модули (пакеты)
Поддержка сети. Протоколы Internet	cgi, Cookie, urllib, uriparse, httplib, smtplib, poplib, telnetlib, socket, asyncore (SocketServer, BaseHTTPServer, xmlrpclib, asynchat)
Поддержка Internet. Форматы данных	quopri, uu, base64, binhex, binascii, rfc822, mimetools, MimeWdter, multifile, mailbox, (email)
Python о себе	parser, symbol, token, keyword, inspect, tokenize, pyclbr, py_compile, compileall, dis, compiler

Модули могут содержать один или несколько классов, с помощью которых создается объект нужного типа, а затем речь идет уже не об именах из модуля, а об атрибутах этого объекта. И наоборот, некоторые модули содержат общие функции для работы над произвольными объектами.

19.1. Модуль sys

Модуль sys содержит информацию о среде выполнения программы интерпретатора Python. Ниже представлены наиболее популярные объекты из этого модуля, остальное можно изучить по документации:

exit([c]) – выход из программы, можно передать числовой код завершения;

0 – в случае успешного завершения, другие числа при аварийном завершении программы;

argv – список аргументов командной строки. Обычно sys.argv[0] содержит имя запущенной программы, а остальные параметры передаются из командной строки;

platform – платформа, на которой работает интерпретатор

version – версия интерпретатора;

setrecursionlimit(limit) – установка уровня максимальной вложенности рекурсивных вызовов;

exc_info() – информация об обрабатываемом исключении.

19.2. Модуль copy

Модуль содержит функции для копирования объектов. Следующий пример:

```
lst1 = [0, 0, 0]
lst = [lst1] *3
print lst[0][1] = 1
print lst
```

Список lst содержит ссылки на один и тот же список. Для того чтобы действительно размножить список, необходимо применить функцию copy() из модуля copy:

```
from copy import copy
lst1 = [0, 0, 0]
lst = [copy(lst1) for i in range(3)]
print lst
lst[0][1] = 1
print lst
```

Получили тот результат, который ожидался:

```
[[0, 0, 0], [0, 0, 0], [0, 0, 0]]
[[0, 1, 0], [0, 0, 0], [0, 0, 0]]
```

В модуле copy есть еще и функция deepcopy() для глубокого копирования, при которой объекты копируются на всю возможную глубину рекурсивно.

19.3. Модули math и cmath

В этих модулях собраны математические функции для действительных и комплексных аргументов. Это те же функции, что используются в языке C. В таблице даны функции модуля math. Там, где аргумент обозначен буквой z, аналогичная функция определена и в модуле cmath.

Функция или константа	Описание
acos(z)	Арккосинус z
asin(z)	Арксинус z
atan(z)	Арктангенс z
atan2(y,x)	atan(y/x)
ceil(x)	Наименьшее целое, большее или равное x
cos(z)	Косинус z
cosh(x)	Гиперболический косинус x
e	Константа e
exp(z)	Экспонента (то есть e * * z)
fabs(x)	Абсолютное значение x
floor(x)	Наибольшее целое, меньшее или равное x
fmod(x, y)	Остаток от деления x на y
frexp(x)	Возвращает мантиссу и порядок x как пару (m, i), где m – число с плавающей точкой, а i – целое, такое что x = m * 2. **i. Если 0 – возвращает (0,0), иначе 0.5<=abs(m)<1.0
hypot(x,y)	sqrt(x*x+y*y)
ldexp(m,i)	m * (2**i)
log(z)	Натуральный логарифм z
log10(z)	Десятичный логарифм z
modf(x)	Возвращает пару (y, q) – дробную и целую часть x. Обе части имеют знак исходного числа
pi	Константа Пи
pow(x,y)	x**y
sin(z)	Синус z
sinh(z)	Гиперболический синус z
sqrt (z)	Корень квадратный от z
tan(z)	Тангенс z
tanh(z)	Гиперболический тангенс z

19.4. Модуль random

Этот модуль генерирует псевдослучайные числа для нескольких различных распределений. Наиболее используемые функции:

Функция	Описание
random()	Генерирует псевдослучайное число из полуоткрытого диапазона [0.0,1.0)
choice(s)	Выбирает случайный элемент из последовательности s
shuffle(s)	Размешивает элементы изменчивой последовательности s на месте
randrange([start,] stop[, step])	Выдает случайное целое число из диапазона
range (start, stop,step)	Аналогично choice (range (start, stop, step))
normalvariate(mu, sigma)	Число из последовательности нормально распределенных псевдослучайных чисел, mu – среднее, sigma – среднеквадратическое отклонение (sigma > 0)

В модуле есть функция seed(n), которая позволяет установить генератор случайных чисел в некоторое состояние, например если возникнет необходимость многократного использования одной и той же последовательности псевдослучайных чисел.

19.5. Модуль sets

Модуль реализует тип данных для множеств. Следующий пример показывает, как использовать этот модуль. Следует заметить, что в Python 2.4 и старше тип set стал встроенным:

```
import sets
A = sets.Set([1, 2, 3])
B = sets.Set([2, 3, 4])
print A|B, A&B, A-B, A^B
for i in A:
    if i in B:
        print i,
```

В результате будет выведено:

```
Set([1, 2, 3, 4])
Set([2, 3])
Set([1])
Set([1, 4])
2 3
```

19.6. Модуль locale

Модуль locale применяется для работы с культурной средой, где могут использоваться свои правила написания чисел, валют, времени и даты и т. п.

19.7. Модуль os

Разделители каталогов и другие связанные с этим обозначения доступны в виде констант.

Константа	Значение
os.curdir	Текущий каталог
os.pardir	Родительский каталог
Константа	Значение
os.sep	Разделитель элементов пути
os.altsep	Другой разделитель элементов пути
os.pathsep	Разделитель путей в списке путей
os.defpath	Список путей по умолчанию
os.linesep	Признак окончания строки

Программа на Python работает в операционной системе в виде отдельного процесса. Функции модуля os дают доступ к различным значениям, относящимся к процессу и к среде, в которой он исполняется. Одним из важных объектов, доступных из модуля os, является словарь environ. Например, с помощью переменных окружения web-сервер передает некоторые параметры в CGI-сценарий. В следующем примере можно получить переменную окружения path:

```
import os
PATH = os.environ['PATH']
```

Группа функций посвящена работе с файлами и каталогами. Приводятся только те функции, которые доступны как в Unix, так и в Windows. Режим запрашиваемого доступа указывается значением flags, составленных комбинацией (побитовым ИЛИ) флагов.

Функция	Действие
access(path, flags)	Доступность файла или каталога с именем path
os.F_OK	Файл существует
os.R_OK	Из файла можно читать
os.W_OK	В файл можно писать
os.X_OK	Файл можно исполнять, каталог можно просматривать
chdir(path)	Делает path текущим рабочим каталогом
getcwd()	Текущий рабочий каталог
chmod(path, mode)	Устанавливает режим доступа к path в значение mode

Режим доступа можно получить, скомбинировав флаги (см. ниже). Функция chmod() не дополняет действующий режим, а устанавливает его заново.

Функция	Действие
listdir(dir)	Возвращает список файлов в каталоге dir
mkdir(path[, mode])	Создает каталог path
makedirs(path[, mode])	Аналог mkdir(), создающий все необходимые каталоги, если они не существуют. Возбуждает исключение, когда последний каталог уже существует

Окончание таблицы

Функция	Действие
remove(path), unlink(path)	Удаляет файл path
rmdir(), removedirs()	Удаление каталогов. Удаляет path до первого непустого каталога. В случае, если самый последний вложенный подкаталог в указанном пути не пустой, возбуждается исключение OSError
rename(src, dst)	Переименовывает файл или каталог src в dst
renames(src, dst)	Аналог rename(), создающий все необходимые каталоги для пути dst и удаляющий пустые каталоги пути src

19.8. Модуль shelve

Для хранения объектов в родном для Python формате применяется shelve. Интерфейс shelve такой же, как у словаря. Модуль shelve позволяет создать объектно-ориентированную базу данных, обеспечивающую сохраняемость объектов, «сериализуемых» [10] с использованием модуля pickle и anydbm. Файлы, создаваемые этим модулем, представлены в двоичном формате.

Функция	Действие
import shelve	Подключение модуля
data = ("abc", 12)	Организация данных
key = "key"	Ключ (строка)
filename = "polka.dat"	Имя файла для хранения
d=shelve.open(filename)	Открытие файла
d[key] = data	Сохранить данные с ключом key
data= d[key]	Загрузить значение по ключу из базы
len(d)	Получить количество объектов в файле

Окончание таблицы

Функция	Действие
d.sync()	Запись изменений в БД на диске
del d[key]	Удалить ключ и значение
flag = d.has_key(key)	Проверка наличия ключа
lst = d.keys()	Список ключей
d.close()	Закрытие полки

На базе модуля shelve создадим свой модуль.

```
from shelve import *
def put_shelve(key,data,fname):
        d = open(fname)
        d[key] = data
        d.sync()
        d.close()
        return 1
def get_shelve(key,fname):
        d = open(fname)
        if d.has_key(key):
                data=d[key]
                d.close()
         return data
        else:
                d.close()
                return 0
def del_shelve(key,fname):
        d = open(fname)
        if d.has_key(key):
                del d[key]
                d.sync()
                d.close()
                return 1
        else:
                d.close()
                return 0
```

```
a={'10':('name1','age1'),'111':('2','3'),'121':('1','2')}
put_shelve('key',a,'data.dat')
b=get_shelve('key','data.dat')
print b
if del_shelve('key','data.dat'): print 'Данные удалены'
```

19.9. Модуль csv

Формат CSV (comma separated values – значения, разделенные запятыми) достаточно популярен для обмена данными между электронными таблицами и базами данных. Следующий ниже пример посвящен записи в CSV-файл и чтению из него:

```
mydata = [(1, 2, 3) , (1, 3, 4)]
import csv
## Запись в файл:
f = file("my.csv", "w")
writer = csv.writer(f)
for row in mydata:
        writer.writerow(row)
f.close()
#Чтение из файла:
reader = csv.reader(file("my.csv"))
for row in reader:
        print row
```

В данном случае количество рекурсивных вызовов растет экспоненциально от числа n, что совсем не соответствует временной сложности решаемой задачи.

При работе с рекурсивными функциями можно легко превысить глубину допустимой в Python рекурсии. Для настройки глубины рекурсии следует использовать функцию setrecursionlimit(N) из модуля sys, установив требуемое значение N.

20. ФУНКЦИИ КАК ПАРАМЕТРЫ И РЕЗУЛЬТАТ

Функции являются такими же объектами Python, как числа, строки или списки. Это означает, что их можно передавать в качестве параметров функций или возвращать из функций.

Функции, принимающие в качестве аргументов или возвращающие другие функции в результате их выполнения, называют функциями высшего порядка. В большинстве случаев таким образом строится механизм обратных вызовов (callbacks), но встречаются и другие варианты. Например, алгоритм поиска может вызывать переданную ему функцию для каждого найденного объекта.

20.1. Функция apply()

Функция apply() применяет функцию, переданную в качестве первого аргумента, к параметрам, которые переданы вторым и третьим аргументом. Эта функция в Python устарела, так как вызвать функцию можно с помощью обычного синтаксиса вызова функции. Позиционные и именованные параметры можно передать с использованием звездочек:

```
>>> lst = [1, 2, 3]
>>> dct= {'a' : 4, 'b' : 5}
>>> apply(max, lst)
3
>>> apply(dict, [], dct)
{'a': 4, 'b': 5}
```

20.2. Обработка последовательностей

Многие алгоритмы сводятся к обработке массивов данных и получению новых массивов данных в результате. Среди встроенных функций Python есть несколько для работы с последовательностями.

Под последовательностью в Python понимается любой тип данных, который поддерживает интерфейс последовательности (это несколько специальных методов, реализующих операции над последовательностями, которые в данном курсе обсуждаться не будут).

Следует заметить, что тип, основной задачей которого является хранение, манипулирование и обеспечение доступа к самостоятельным данным, называется контейнерным типом или просто контейнером. Примеры контейнеров в Python – списки, словари.

20.2.1. Функции range() и xrange()

Функция range() уже упоминалась при рассмотрении цикла for. Эта функция принимает от одного до трех аргументов. Если аргумент

всего один, то она генерирует список чисел от 0 (включительно) до заданного числа (исключительно). Если аргументов два, то список начинается с числа, указанного первым аргументом. Если аргументов три, то третий аргумент задает шаг.

```
>>> print range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> print range(1, 10)
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> print range(1, 10, 3)
[1, 4, 7]
```

Функция xrange() – аналог range(), более предпочтительный для использования при последовательном доступе, например в цикле for или с итераторами. Она возвращает специальный xrange-объект, который ведет себя почти как список, порождаемый range(), но не хранит в памяти все выдаваемые элементы.

20.2.2. Функция map()

Для применения некоторой функции ко всем элементам последовательности применяется функция map(f, *args). Первый параметр этой функции – функция, которая будет применяться ко всем элементам последовательности. Каждый следующий ($n + 1$)-й параметр должен быть последовательностью, так как каждый его элемент будет использован в качестве n-го параметра при вызове функции f(). В итоге получим список, составленный из результатов выполнения этой функции.

В следующем примере складываются значения из двух списков:

```
>>> l1 = [2, 7, 5, 3]
>>> l2 = [-2, 1, 0, 4]
>>> print map(lambda x,y: x+y, l1, l2)
 [0, 8, 5, 7]
```

В этом примере применена безымянная функция для получения суммы двух операндов ко всем элементам l1 и l2. В случае, если одна из последовательностей короче другой, вместо соответствующего операнда будет использоватться None, что, конечно, собьет операцию

сложения. В зависимости от решаемой задачи можно либо видоизменить функцию, либо считать разные по длине последовательности ошибкой, которую нужно обрабатывать как отдельную ветвь алгоритма.

Частный случай применения map() – использование None в качестве первого аргумента. В этом случае просто формируется список кортежей из элементов исходных последовательностей:

```
>>> l1 = [2, 7, 5, 3]
>>> l2 = [-2, 1, 0, 4]
>>> print map(None, l1, l2)
[(2, -2), (7, 1), (5, 0) , (3, 4)]
```

20.2.3. Функция filter()

Другой часто встречающейся операцией является фильтрование исходной последовательности в соответствии с некоторым предикатом (условием). Функция filter(f,seq) принимает два аргумента: функцию с условием и последовательность, из которой берутся значения. В результирующую последовательность попадут только те значения из исходной, для которой f() возвратит истину. Если в качестве f задано значение None, результирующая последовательность будет состоять из тех значений исходной, которые имеют истинностное значение True. Например, в следующем фрагменте кода можно избавиться от символов, которые не являются буквами:

```
>>>filter(lambda x: x.isalpha(),'Hi,there! I am eating an apple.')
' Hitherelameatinganapple'
```

20.2.4. Списковые включения

Для более естественной записи обработки списков в Python была внесена новинка – списковые включения. Фактически это специальный сокращенный синтаксис для вложенных циклов, for и условий if, внутри которых определенное выражение добавляется к списку, например:

```
all_pairs = []
for i in range(5) :
    for j in range(5) :
        if i <= j: all_pairs.append((i, j))
```

Все это можно записать в виде спискового включения так:

```
all_pairs = [(i, j) for i in range (5) for j in range (5) if i <= j]
```

Как легко заметить, списковые включения позволяют заменить map() и filter() на более удобные для прочтения конструкции.

В нижеприведенной таблице даны эквивалентные выражения в разных формах.

В форме функции	В форме спискового включения
filter(f, lst)	[x for x in 1st if f(x)]
filter(None, lst)	[x for x in 1st if x]
map(f, lst)	[f(x) for x in lst]

20.2.5. Функция sum()

Получить сумму элементов можно с помощью функции sum ():

```
>>> sum(range(10))
45
```

Эта функция работает только для числовых типов, она не может конкатенировать строки. Для конкатенации списка строк следует использовать метод join().

```
>>>L=['Мама ','мыла ','раму.']
>>>join(L)
'Мама мыла раму.'
```

20.2.6. Функция reduce()

Для организации цепочечных вычислений (вычислений с накоплением результата) можно применять функцию reduce(), которая принимает три аргумента: функцию двух аргументов, последовательность и начальное значение. С помощью этой функции можно, в частности, реализовать функцию sum():

```
def sum(lst, start):
    return reduce(lambda x, y: x + y, lst, start)
```

В качестве передаваемого объекта может оказаться список, который позволит накапливать промежуточные результаты. Тем самым reduce() может использоваться для порождения последовательностей.

```
lst = range(10)
f = lambda x, y: (x[0] + y, x[1]+[x[0] + y])
print reduce(f, lst, (0, []))
В итоге получается:
(45, [0, 1, 3, 6, 10, 15, 21, 28, 36, 45])
```

20.2.7. Функция zip()

Функция zip() возвращает список кортежей, в котором 1-й кортеж содержит *i*-е элементы аргументов-последовательностей. Длина результирующей последовательности равна длине самой короткой из последовательностей аргументов:

```
>>> print zip(range(5), "abcde")
[(0, 'a'), (1,'b'), (2, 'c'), (3, 'd'), (4, 'e')]
```

20.2.8. Итераторы

Применять для обработки данных явные последовательности не всегда эффективно, так как на хранение временных данных может тратиться много оперативной памяти. Более эффективным решением представляется использование итераторов – специальных объектов, обеспечивающих последовательный доступ к данным контейнера. Если в выражении есть операции с итераторами вместо контейнеров, то промежуточные данные не будут требовать много места для хранения – они запрашиваются по мере необходимости для вычислений. При обработке данных с использованием итераторов память будет требоваться только для исходных данных и результата, да и то необязательно вся сразу – ведь данные могут читаться и записываться в файл на диске.

Итераторы можно применять вместо последовательности в операторе for. Более того, оператор for запрашивает у последовательно-

сти ее итератор. Объект файлового типа – тоже (построчный) итератор, что позволяет обрабатывать большие файлы, не считывая их целиком в память.

Использовать итератор можно и вручную. Любой объект, поддерживающий интерфейс итератора, имеет метод next(), который при каждом вызове выдает очередное значение итератора. Если больше значений нет, то возбуждается исключение StopIteration. Для получения итератора по некоторому объекту необходимо прежде применить к этому объекту функцию iter() (цикл for делает это автоматически).

В Python имеется модуль itertools, который содержит набор функций, комбинируя которые, можно составлять достаточно сложные схемы обработки данных с помощью итераторов. Далее рассматриваются некоторые функции этого модуля.

20.2.9. Функция iter()

Функция iter() имеет два варианта использования. В первом она принимает всего один аргумент, который должен «уметь» предоставлять свой итератор. Во втором один из аргументов – функция без аргументов, другой – стоповое значение. Итератор вызывает указанную функцию до тех пор, пока та не возвратит стоповое значение. Второй вариант встречается много реже первого и обычно внутри метода класса, так как сложно порождать значения «на пустом месте»:

```
it1 = iter([1, 2, 3, 4, 5])
def forit(mystate=[]):
    if len(mystate)<3: mystate.append(" ")
    return " "
it2 = iter(forit,None)
print[x for x in it1]
print [x for x in it2]
```

Если функция не возвращает значения явно, то она возвращает None, что и использовано в примере выше.

20.2.10. Функция enumerate()

Функция enumerate() создает итератор, нумерующий элементы другого итератора. Результирующий итератор выдает кортежи, в ко-

торых первый элемент – номер (начиная с нуля), а второй – элемент исходной последовательности:

```
>>> print [x for x in enumerate("abcd")]
[(0, 'a'), (1, 'b'), (2, 'c'), (3, 'd')]
```

20.2.11. Функция sorted()

Функция sorted(), появившаяся в версии Python 2.4, позволяет создавать итератор, выполняющий сортировку:

```
>>> sorted('avdsdf)
['a' , 'd' , 'd' , 'f, 's' , 'v']
```

20.2.12. Модуль itertools

Функция itertools.chain() позволяет сделать итератор, состоящий из нескольких соединенных последовательно итераторов. Итераторы задаются в виде отдельных аргументов:

```
    from itertools import chain
    it1 = iter([1,2,3])
    it2 = iter([8,9,0])
for i in chain(it1, it2): print i, # даст 123890
```

Функция itertools.repeat() строит итератор, повторяющий некоторый объект заданное количество раз:

```
>>>for i in itertools.repeat(1,4): print i,
1111
```

Бесконечный итератор itertools.count() используется так:

```
for i in itertools.count(1):
        print i,
        if  i > 100: break
```

Можно бесконечно повторять и некоторую последовательность (или значения другого итератора) с помощью функции itertools.cycle ():

```
tango = [1, 2, 3]
for i in itertools.cycle(tango): print i,
```

Аналогами map() и filter() в модуле itertools являются itertools.imap() и itertools.ifilter(). Отличие itertools.imap() от map() заключается в том, что вместо значения от преждевременно завершившихся итераторов объект None не подставляется:

```
>>>for i in map(lambda x,y:(x,y),[1,2],[1,2,3]): print i,
(1,1) (2,2) (None,3)
>>>from itertools import imap
>>>for i in imap(lambda x,y:(x,y),[1,2],[1,2,3]): print i,
(1, 1) (2, 2)
```

Здесь следует заметить, что обычная функция map() нормально воспринимает итераторы в любом сочетании с итерабельными (поддающимися итерациям) объектами:

```
>>>for i in map(lambda x, y: (x,y) , iter([1,2]),[1,2,3]): print i,
(1, 1) (2, 2) (None, 3)
```

Функция itertools.starmap() подобна itertools.imap(), но имеет всего два аргумента. Второй аргумент – последовательность кортежей, каждый кортеж которой задает набор параметров для функции (первого аргумента):

```
>>> from itertools import starmap
>>> for i in starmap(lambda x, y: str(x) +y, [(1,'a'),(2,'b')]): print i,
la 2b
```

Функция ifilter() работает как filter(). Кроме того, в модуле itertools есть функция ifilterfalse(), которая добавляет отрицание к значению функции:

```
>>>for i in ifilterfalse(lambda x: x>0,[1,-2,3,-3]):  print i,
-2 -3
```

Некоторую новизну вносит другой вид фильтра itertools.takewhile() и его «отрицательный» аналог itertools.dropwhile(). Следующий пример поясняет их принцип действия:

```
for i in takewhile(lambda x: x > 0, [1, -2, 3, -3]): print i,
print for i in dropwhile(lambda x: x >0,[1,-2,3,-3]): print i,
```

Обобщенная природа функций Python и полиморфизм, не завязанный целиком на наследовании, – вот свойства языка Python, позволяющие иметь большую гибкость в комбинации процедурного и объектно-ориентированного подходов.

21. МАТРИЧНЫЕ ВЫЧИСЛЕНИЯ

Здесь приводится обзор других пакетов для научных вычислений.

Numeric Python – это несколько модулей для вычислений с многомерными массивами, необходимых для многих численных приложений. Модуль Numeric вносит в Python возможности таких пакетов и систем, как MatLab, Octave(аналог MatLab), APL, J, S+, IDL. Стоит заметить, что некоторые синтаксические возможности Python, связанные с использованием срезов, были специально разработаны для Numeric.

Numeric Python имеет средства для:

- матричных вычислений LinearAlgebra;
- быстрого преобразования Фурье FFT;
- работы с недостающими экспериментальными данными MA;
- статистического моделирования RNG;
- эмуляции базовых функций программы MatLab.

21.1. Модуль Numeric

Модуль Numeric определяет полноценный тип-массив и содержит большое число функций для операций с массивами. *Массив* – это набор однородных элементов, доступных по индексам. Массивы модуля Numeric могут быть многомерными, то есть иметь более одной *размерности***.**

21.2. Создание массива

Для создания массива можно использовать функцию array(), в которой указано содержимое массива (в виде вложенных списков) и типа. Функция array() делает копию, если ее аргумент – массив. Функция asarray() работает аналогично, но не создает нового массива, когда ее аргумент уже является массивом:

```
>>> from Numeric import *
>>> print array([[1, 2], [3, 4], [5, 6]])
[[1 2]
[3 4]
[5 6]]
>>> print array([[l, 2, 3], [4, 5, 6]], Float)
[[ 1. 2. 3.]
[ 4.  5.  6.]]
>>> print array([78, 85, 77, 69, 82, 73, 67], 'c')
[NUMERIC]
```

В качестве элементов массива можно использовать следующие типы: Int8-Int32, UnsignedInt8-UnsignedInt32, Float8-Float64, Complex8-Complex64 и PyObject. Числа 8, 16, 32 и 64 показывают количество битов для хранения величины. Типы Int, UnsignedInteger, Float и Complex соответствуют наибольшим принятым на данной платформе значениям. В массиве можно также хранить ссылки на произвольные объекты.

Количество размерностей и длина массива по каждой оси называются формой массива (shape). Доступ к форме массива реализуется через атрибут shape:

```
>>> from Numeric import *
>>> a = array(range(15), Int)
>>> print a.shape(15,)
>>> print a
[0 1 2 3 4 5 6 7 8 9 10 11 12 13 14]
>>> a.shape = (3, 5)
>>> print a.shape
(3, 5)
>>> print a
```

```
[[01234]
 [56789]
 [10 11 12 13 14]]
```

21.3. Методы массивов

Придать нужную форму массиву можно функцией Numeric, reshape(). Эта функция сразу создает объект-массив нужной формы из последовательности.

```
>>> import Numeric
>>> print Numeric.reshape("абракадабр", (5, -1))
[[а б]
 [р а]
 [к а]
 [д а]
 [б р]]
```

В этом примере цифра -1 в указании формы массива говорит о том, что соответствующее значение можно вычислить. Общее количество элементов массива известно (десять), поэтому длину вдоль одной из размерностей задавать не обязательно. Через атрибут flat можно получить одномерное представление массива:

```
>>>a =array([[1, 2], [3, 4]])
>>> b = a.flat
>>> b
array([1, 2, 3, 4])
>>> b[0] = 9
>>> b
array([9, 2, 3, 4])
>>> a array([[9, 2], [3, 4]])
```

Следует заметить, что это новый вид того же массива, поэтому присваивание значений его элементам приводит к изменениям в исходном массиве. Функция Numeric.resize() похожа на Numeric.reshape(), но может подстраивать число элементов:

```
>>> print Numeric.resize("NUMERIC", (3, 2))
[[N U]
[M E]
[R 111
>>> print Numeric.resize("NUMERIC", (3, 4))
[[N U M E]
 [R I C N]
 [U M E R]]
```

Функция Numeric.zeros() порождает массив из одних нулей, а Numeric.ones() – из одних единиц. Единичную матрицу можно получить с помощью функции Numeric. identity(n):

```
>>> print Numeric.zeros((2,3))
[[0 0 0]
[0 0 0]]
>>> print Numeric.ones((2,3))
[[1 1 1]
[1 1 1]]
>>>print Numeric.identity(4)
[[1 000]
[0100]
[0010]
[000 1]]
```

Для копирования массивов можно использовать метод copy():

```
>>> import Numeric
>>> a = Numeric.arrayrange(9)
>>> a.shape = (3, 3)
>>> print a
[[0 1 2]
[3 4 5]
[6 7 8]]
>>> a1  = a.copy()
>>> a1[0, 1] = -1   #операция над копией
>>> print a
[[0 1 2]
```

```
[3 4 5]
[6 7 8]]
```

Массив превращается обратно в список методом tolist():

```
>>> a.tolist()
[[0, 1, 2], [3, 4, 5], [6, 7, 8]]
```

21.4. Срезы массивов

Объекты-массивы Numeric используют расширенный синтаксис выделения среза. Следующие примеры иллюстрируют различные варианты записи срезов. Функция Numeric.arrayrange() является аналогом range() для массивов.

```
>>> import Numeric
>>> a = Numeric.arrayrange(24) + 1
>>> a.shape = (4, 6)
>>> print a          #исходный массив
[[123456]
[7 8 9 10 11 12]
[13 14 15 16 17 18]
[19 20 21 22 23 24]]
>>> print a[1,2]   # элемент
1,2 9
>>> print a[l,:]   # строка 1
[7 8 9 10 11 12]
>>> print a[l]            # тоже строка 1
[7 8 9 10 11 12]
>>> print a[:,l]   # столбец 1
[ 2 8 14 20]
>>> print a[-2,:]  #предпоследняя строка
[13 14 15 16 17 18]
>>> print a[0:2,1:3]   #окно 2x2
[[2 3]
[8 9]]
 >>> print a[l,::3] #каждый третий элемент строки 1
[7 10]
>>> print a[:,::-!]#элементы строк в обратном порядке
```

```
[[6 5 4 3 2 1]
[12 11 10 9 8 7]
[18 17 16 15 14 13]
[24 23 22 21 20 19]]
```

Срез не копирует массив (как это имеет место со списками), а дает доступ к некоторой части массива. Далее в примере меняется на 0 каждый третий элемент строки 1:

```
>>>a[1,::3]=Numeric.array([0,0])
>>> print a
[[1 2 3 4 5 6]
[0 8 9 0 11 121
[13 14 15 16 17 18]
[19 20 21 22 23 24]]
```

В следующих примерах находит применение достаточно редкая синтаксическая конструкция: срез с многоточием (Ellipsis). Многоточие ставится для указания произвольного числа пропущенных размерностей (:, :, ..., :):

```
>>> import Numeric
>>> a = Numeric.arrayrange(24) + 1
>>> a.shape = (2,2,2,3)
>>> print a
[[[[ 1 2 3]
   [4 5 6]]
[[7 8 9]
 [10 11 12]]]
[[[13 14 15]
  [16 17 18]]
 [[19 20 21]
  [22 23 24]]]]
>>>print a[0, ...]    # 0-й блок
[[[ 1 2 3]
  [4 5 6]]
 [[7  8 9]
  [10 11 12]]]
```

```
>>>print a[0, :, :,0] #по перв.и послед.размерности
[[1 4]
[ 7 10]]
>>> print a[0, ...,0]        #то же,с использ.многоточия
[[ 1 4]
[ 7 10]]
```

21.5. Универсальные функции элементов массивов

Модуль Numeric определяет набор функций для применения к элементам массива. Функции применимы не только к массивам, но и к последовательностям (к сожалению, итераторы пока не поддерживаются). В результате получаются массивы.

Функция	Описание
1	2
add(x,y)	Сложение
subtract(x, y)	Вычитание
multiply(x, y)	Умножение
divide(x, y)	Деление
remainder(x,y),fmod(x,y)	Получение остатка от деления (для целых чисел и чисел с плавающей запятой)
power(x)	Возведение в степень
sqrt(x)	Извлечение корня квадратного
negative(x), absolute(x), fabs(x)	Смена знака и абсолютное значение
ceil(x), floor(x)	Наименьшее (наибольшее) целое, большее (меньшее) или равное аргументу
hypot(x,y)	Длина гипотенузы по двум катетам: x,y
sin(x), cos(x), tan(x)	Тригонометрические функции
arcsin(x), arccos(x), arctan(x)	Обратные тригонометрические функции

Окончание таблицы

1	2
arctan2(x,y)	Арктангенс от частного аргумента
sinh(x), cosh(x), tanh(x)	Гиперболические функции
arcsinh(x), arccosh(x), arctanh(x)	Обратные гиперболические функции
exp(x)	Экспонента
log(x) , logl0(x)	Натуральный и десятичный логарифмы
maximum (x, y)	Максимум
minimum(x,y)	Минимум
conjugate (x)	Сопряжение (для комплексных чисел)
equal(x, y), not_equal(x,y)	Равно, не равно
greater(x, y), greater_equal(x, y)	Больше, больше или равно
less(x,y), less_equal(x, y)	Меньше, меньше или равно
logical_and(x, y), logical_or(x,y)	Логические И, ИЛИ
logical_xor(x,y)	Логическое исключающее ИЛИ
logical_not(x)	Логическое НЕ
bitwise_and(x, y), bitwise_or(x,y)	Побитовые И, ИЛИ
bitwise_xor(x,y)	Побитовое исключающее ИЛИ
invert(x)	Побитовая инверсия
left_shift(x, n), right_shift(x, n)	Побитовые сдвиги влево и вправо на n битов

21.6. Функции модуля Numeric

В таблице представлены функции модуля Numeric, которые являются краткой записью некоторых наиболее употребительных сочетаний функций и методов. Параметр axis в функциях указывает размерность.

Функция	Аналог функции
sum(a, axis)	add.reduce(a, axis)
cumsum(a, axis)	add.accumulate(a, axis)
product(a, axis)	multiply.reduce(a, axis)
cumproduct(a, axis)	multiply.accumulate(a, axis)
alltrue(a, axis)	logical_and.reduce(a, axis)
sometrue(a, axis)	logical_or.reduce(a, axis)

21.7. Модуль LinearAlgebra

Модуль LinearAlgebra содержит алгоритмы линейной алгебры, в частности нахождение определителя матрицы, решений системы линейных уравнений, обращения матрицы, нахождения собственных чисел и собственных векторов матрицы, разложения матрицы на множители.

Функция LinearAlgebra.determinant() находит определитель матрицы:

```
>>> import Numeric, LinearAlgebra
>>> print LinearAlgebra.determinant( Numeric.array([[1,-2],[1,5]]))
7
```

Функция LinearAlgebra.solve_linear_equations() решает линейные уравнения вида ax=b по заданным аргументам a и b:

```
>>> import Numeric, LinearAlgebra
>>> a = Numeric.array([[1.0,2.0],[0.0,1.0]])
>>> b = Numeric.array([1.2,1.5])
>>> x = LinearAlgebra.solve_linear_equations(a,b)
>>> print "x=", x
x = [-1.8 1.5]
```

Когда матрица *a* имеет нулевой определитель, система имеет не единственное решение и возбуждается исключение LinAlgError:

```
>>> a=Numeric.array([[1.0,2.0],[0.5,1.0]])
>>> x=LinearAlgebra.solve_linear_equations(a,b)
Traceback(most recent call last):
File "<stdin>", line 1, in ?
File "/usr/local/lib/python2.3/site-
packages/Numeric/LinearAlgebra.py", line 98, in solve_linear_equations
raise LinAlgError, 'Singular matrix' LinearAlgebra.LinAlgError: Singu-
lar matrix
```

Функция LinearAlgebra.inverse() находит обратную матрицу. Однако не следует решать линейные уравнения умножением LinearAlgebra.inverse() на обратную матрицу, так как она определена через LinearAlgebra.solve_linear_equations():

```
solve_linear_equations(a,Numeric.identity(a.shape[0]))
```

Функция LinearAlgebra.eigenvalues() находит собственные значения матрицы, а LinearAlgebra.eigenvectors() – собственные значения и собственные векторы.

```
>>>from Numeric import array, dot
>>>from LinearAlgebra import *
>>> a=array([[-5,2],[2,-7]])
>>> lmd = eigenvalues(a)
>>> print "Собственные значения:",lmd
Собственные значения: [-3.76393202 -8.23606798]
>>> (lmd, v) = eigenvectors(a)
>>> print "Собственные вектора:"
Собственные вектора:
>>> print v
[[ 0.85065081 0.52573111]
[-0.52573111 0.85065081]]
>>> print "Проверка:", dot(a,v[0]) – v[0] * lmd[0]
Проверка: [ -4.44089210e-16   2.22044605e-16]
```

Проверка показывает, что тождество выполняется с достаточно большой точностью (числа совсем маленькие, практически нули): собственные числа и векторы найдены верно.

21.8. Модуль RandomArray

В модуле собраны функции для генерации массивов случайных чисел различных распределений и свойств. Их можно применять для математического моделирования стохастических систем.

Функция RandomArray.random() создает массивы из псевдослучайных чисел, равномерно распределенных в интервале (0,1):

```
>>>import RandomArray
>>>print RandomArray.random(10)
#10 псевдослучайных чисел
[0.28374212 0.19260929 0.07045474 0.305476820.10842083
0.14049676 0.01347435 0.37043894 0.47362471 0.37673479]
>>> print RandomArray.random([3,3])
# массив 3x3 из псевдослучайных чисел
[[ 0.53493741 0.44636754 0.20466961]
[ 0.8911635   0.03570878 0.00965272]
[ 0.78490953 0.20674807 0.23657821]]
```

Функция RandomArray.randint() для получения массива равномерно распределенных чисел из заданного интервала и заданной формы:

```
>>> pront RandomArray.randint(1,10,[10])
[8 1 9 9 7 5 2 5 3 2]
>>> print RandomArray.randint(1,10,[10])
[2 2 5 5 7 7 3 4 3 7]
```

Можно получать и случайные перестановки с помощью RandomArray.permutation():

```
>>> print RandomArray.permutation(6)
[4 0 1 3 2 5]
>>> print RandomArray.permutation(6)
[1 2 0 3 5 4]
```

Доступны и другие распределения для получения массива нормально распределенных величин с заданным средним и стандартным отклонением:

```
>>> print RandomArray.normal(0,1,30)
[-1.0944078   1.24862444 0.20415567 -0.74283403 0.72461408 -
0.57834256 0.30957144 0.8682853 1.10942173 -0.39661118
1.33383882 1.54818618 0.18814971 0.89728773 -0.86146659
0.0184834 -1.46222591 -0.78427434 1.09295738 -1.09731364
1.34913492 -0.75001568 -0.11239344 2.73692131 -0.19881676 -
0.49245331 1.54091263 -1.81212211 0.46522358 -0.08338884]
```

Следующая таблица приводит функции для других распределений.

Функция и ее аргументы	Описание
1	2
F(dfn,dfd,shape=[])	F-распределение
beta(a,b,shape=[])	Бета-распределение
binomial(trials,p,shape=[])	Биномиальное распределение
chi_square(df,shape=[])	Распределение хи-квадрат
exponential(mean, shape=[])	Экспоненциальное распределение
gamma(a,r,shape=[])	Гамма-распределение
multivariate_normal(mean,cov,shape=[])	Многомерное нормальное распределение
negative_binomial(trials,p, shape=[])	Негативное биномиальное
noncentral_F(dfn,dfd,nconc, shape=[])	Нецентральное F-распределение
noncentral_chi_square(df, nconc, shape=[])	Нецентральное хи-квадрат распределение
normal(mean,std,shape=[])	Нормальное распределение
permutation(n)	Случайная перестановка
poisson(mean,shape=[])	Пуассоновское распределение
randint(min,max=None,shape=[])	Случайное целое

Окончание таблицы

1	2
random(shape=[])	Равномерное распределение на интервале (0, 1)
random_integers(max,min=l, shape=[])	Случайное целое
standard_normal(shape=[])	Стандартное нормальное распределение
uniform(min,max,shape=[])	Равномерное распределение

22. ОБРАБОТКА ТЕКСТОВ. РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ. UNICODE

Под обработкой текстов понимается анализ, преобразование, поиск, порождение текстовой информации. Обработка естественных текстов – это процесс неискусственного интеллекта. Здесь не рассматривается обработка текстов посредством текстовых процессоров и редакторов, хотя некоторые из них (например, Cooledit) предоставляют возможность писать макрокоманды на Python.

Для Python созданы модули для работы с естественными языками, для лингвистических исследований. Хорошим примером служит nltk (the Natural Language Toolkit).

22.1. Строки

Строки в языке Python являются типом данных, специально предназначенным для обработки текстовой информации. Строка может содержать произвольно длинный текст (ограниченный имеющейся памятью).

В новых версиях Python имеются два типа строк: обычные строки (последовательность байтов) и Unicode-строки (последовательность символов). В Unicode-строке каждый символ может занимать в памяти 2 или 4 байта в зависимости от настроек периода компиляции. Четырехбайтовые знаки используются в основном для восточных языков.

В языке Python и стандартной библиотеке за некоторыми исключениями строки и Unicode-строки взаимозаменяемы, в собственных приложениях для совместимости с обоими видами строк следует

избегать проверок на тип. Если это необходимо, можно проверять принадлежность базовому (для строк и Unicode-строк) типу с помощью isinstance(s,basestring).

При использовании Unicode-строк следует помнить, что именно Unicode-представление является главным, а все остальные кодировки – лишь частными случаями представления текста, которые не могут передать всех символов. Без такой установки будет непонятно, почему преобразование из восьмибитной кодировки называется decode (декодирование). Для внешнего представления можно с успехом использовать кодировку UTF-8, хотя, конечно, это зависит от решаемых задач.

Для того чтобы Unicode–литералы в Python-программе воспринимались интерпретатором правильно, необходимо указать кодировку в начале программы, записав в первой или второй строке, например, следующее (для Unix/Linux):

```
# -*- coding: koi8-r -*-
```

или (под Windows):

```
# -*- coding: cpl251 -*-
```

Могут быть и другие варианты:

```
# -*- coding: latin-1 -*-
# -*- coding: utf-8 -*-
# -*- coding: mac-cyrillic -*-
# -*- coding: iso-5 -*-
```

Полный перечень кодировок (и их псевдонимов):

```
>>> import encodings.aliases
>>> print encodings.aliases.aliases
{'iso_ir_6': 'ascii',
'maccyrillic': 'mac_cyrillic',
'iso_celtic': 'iso8859_14',
 'ebcdic_cp_wt': 'cp037',
'ibm500': 'cp500', ...
```

Если кодировка не указана, то считается, что используется us-ascii. При этом интерпретатор Python будет выдавать предупреждения при запуске модуля:

```
sys:1: DeprecationWarning: Non-ASCII character '\xf0' in file example.py
on line 2, but no encoding declared;
see http://www.python.org/peps/pep-0263.html for details
```

22.2. Строковые литералы

Строки можно задать в программе с помощью строковых литералов. Литералы записываются с использованием апострофов «'», кавычек «"» или этих же символов, взятых трижды. Внутри литералов обратная косая черта имеет специальное значение. Она служит для ввода специальных символов и для указания символов через коды. Если перед строковым литералом поставлено r, то обратная косая черта не имеет специального значения (r – от английского слова raw, строка задается как есть). Unicode-литералы задаются с префиксом s. Вот несколько примеров:

```
s1 = "строка 1"
s2 = r'\1\2'
s3 = """apple\ntree"""    # \n- символ перевода строки
```

Обратная косая черта не должна быть последним символом в литерале, то есть str\ вызовет синтаксическую ошибку.

Указание кодировки позволяет применять в Unicode-литералах приведенную в начале программы кодировку. Если кодировка не указана, можно пользоваться только кодами символов, заданными через обратную косую черту.

22.3. Операции над строками

К операциям над строками, которые имеют специальную синтаксическую поддержку в языке, относятся, в частности, конкатенация (склеивание) строк, повторение строки, форматирование:

```
>>> print"A" +"B","A"*5,"%s" %"A"
AB AAAAA A
```

В операции форматирования левый операнд является строкой формата, а правый может быть либо кортежем, либо словарем, либо некоторым значением другого типа:

```
>>> print "%i" % 234
234
>>> print "%i %s %3.2f" % (5, "ABC", 23.45678)
5 ABC 23.46
>>> a = 123
>>> b = [1, 2, 3]
>>> print "%(a)i: %(b)s" % vars()
123: [1, 2, 3]
```

22.4. Операция форматирования

В строке формата кроме текста могут употребляться спецификации, регламентирующие формат выводимого значения. Спецификация имеет синтаксис:

"%" [ключ][флаг*][шир][.точность][длина_типа]спецификатор
ключ: "(" символ за исключением круглых скобок* ")"
флаг: "+" | "-" I пробел I "#" I "0"
шир: (" 1" ... "9")("0" ... "9")* ("*"
точность: ("1" ... " 9")* | " *"
длина_типа: "a" ... "z" I "A" ... "Z"
спецификатор: "a" ... "z" I "A" ... "z" I"%"
Здесь символы обозначают следующее:

- ключ – ключ из словаря;
- флаг – дополнительные свойства преобразования;
- шир – минимальная ширина поля;
- точность – точность (для чисел с плавающей запятой);
- длина_типа – модификатор типа;
- спецификатор – тип представления выводимого объекта**.**

В следующей таблице приведены некоторые наиболее употребительные значения для спецификации форматирования.

Символ	Применение	Обозначение
1	2	3
0	Флаг	Заполнение нулями слева
-	Флаг	Выравнивание по левому краю
+	Флаг	Обязательный вывод знака числа
Пробел	Флаг	Использовать пробел на месте знака числа
d, i	Спецификатор	Знаковое целое

Окончание таблицы

1	2	3
u	Спецификатор	Беззнаковое целое
o	Спецификатор	Восьмеричное беззнаковое целое
x, X	Спецификатор	Шестнадцатеричное беззнаковое целое (со строчными или прописными латинскими буквами)
e, E	Спецификатор	Число с плавающей запятой в формате с экспонентой
f, F	Спецификатор	Число с плавающей запятой
g,G	Спецификатор	Число с плавающей точкой в более коротком написании (автоматически выбирается e или f)
c	Спецификатор	Одиночный символ (целое число или односимвольная строка)
r	Спецификатор	Любой объект, приведенный к строке функцией repr()
s	Спецификатор	Любой объект, приведенный к строке функцией str()
%	Спецификатор	Знак процента. Для задания одиночного процента необходимо записать %%

22.5. Индексы и срезы

Следует напомнить, что строки являются неизменчивыми последовательностями, поэтому к ним можно применять операции взятия элемента по индексу и срезы:

```
>>> s = "транспорт"
>>> print s[0], s[-1]
т т
>>> print s[-4:]
порт
>>> print s[:5]
транс
>>> print s[4:8]
спор
```

При выделении среза нумеруются не символы строки, а промежутки между ними.

22.6. Модуль string

До того как у строк появились методы, для операций над строками применялся модуль string. Приведенный пример демонстрирует, как вместо функции из модуля string использовать метод (кстати, последнее более эффективно):

```
>>> import string
>>> s = "one,two,three"
>>> print string.split(s, ",")
['one', 'two', 'three']
>>> print s.split(",")
['one', 'two', 'three']
```

В Python 2.4 появилась альтернатива использованию операции форматирования – класс Template. Пример:

```
>>>import string
>>>tpl = string.Template("$a + $b = ${c}")
>>>a = 2
>>>b = 3
>>>c = a + b
>>>print tp1.substitute(vars())
2+3=5
>>>del c              # удаляется имя c
>>>print tp1.safe_substitute(vars() )
2 + 3 = $c
>>>print tp1.substitute(vars(), c=a+b)
2+3=5
>>> print tpl.substitute(vars())
Traceback (most recent call last):
File "/home/rnd/tmp/Python-2.4b2/Lib/string.py",
line 172, in substitute return self.pattern.sub(convert, self.template)
File "/home/rnd/tmp/Python-2.4b2/Lib/string.py",
line 162, in convert val=mapping[named] KeyError:'c'
```

Объект-шаблон имеет два основных метода: substitute() и safe_substitute(). Значения для подстановки в шаблон берутся из словаря (vars() содержит словарь со значениями переменных) или из именованных фактических параметров. Если есть неоднозначность в задании ключа, можно использовать фигурные скобки при написании ключа в шаблоне.

22.7. Методы строк

В таблице ниже приведены некоторые наиболее употребительные методы объектов-строк и Unicode-объектов.

Метод	Описание
1	2
center(w)	Центрирует строку в поле длины w
count(sub)	Число вхождений строки sub в строке
encode([enc[,errors]])	Возвращает строку в кодировке enc. Параметр errors может принимать значения "strict" (по умолчанию), "ignore", "replace" или "xmlcharrefrepiace"
endswith(suffix)	Определяет, оканчивается ли строка на suffix
expandtabs([tabsize])	Заменяет символы табуляции на пробелы. По умолчанию tabsize=8
find(sub[,start[,end]])	Возвращает наименьший индекс, с которого начинается вхождение подстроки sub в строку. Параметры start и end ограничивают поиск окном start: end, но возвращаемый индекс соответствует исходной строке. Если подстрока не найдена, возвращается -1
index(sub[,start[,end]])	Аналогично find(), но возбуждает исключение ValueError в случае неудачи
isalnum()	Возвращает true,если строка содержит только буквы и цифры и имеет ненулевую длину. Иначе – False

Продолжение таблицы

1	2
isalpha()	Возвращает true,если строка содержит только буквы и длина ненулевая
isdecimal()	Возвращает true,если строка содержит только десятичные знаки (только для строк Unicode)и длина ненулевая
isdigit()	Возвращает true,если содержит только цифры и длина ненулевая
islower()	Возвращает true,если все буквы строчные (и их более одной), иначе – false
isnumeric()	Возвращает true, если в строке только числовые знаки (только для Unicode)
isspace()	Возвращает true,если строка состоит только из пробельных символов. Для пустой строки возвращается False
join(seq)	Соединение строк из последовательности seq через разделитель, заданный строкой
lower ()	Приводит строку к нижнему регистру букв
lstrip()	Удаляет пробельные символы слева
replace(old,new[,n])	Возвращает копию строки, в которой подстроки old заменены new. Если задан параметр n, то заменяются только первые n вхождений
rstrip()	Удаляет пробельные символы справа
split([sep[,n]])	Возвращает список подстрок, получающихся разбиением строки a разделителем sep. Параметр n определяет максимальное количество разбиений (слева)
startswith(prefix)	Определяет, начинается ли строка с подстроки prefix
strip()	Удаляет пробельные символы в начале и в конце строки

Окончание таблицы

1	2
translate(table)	Производит преобразование с помощью таблицы перекодировки table, содержащей словарь для перевода кодов в коды (или в None,чтобы удалить символ)
translate (table[,dc])	Для Unicode-строк. То же, но для обычных строк. Вместо словаря – строка перекодировки на 256 символов, которую можно сформировать с помощью функций string, maketrans(). Необязательный параметр dc задает строку с символами, которые необходимо удалить
upper()	Переводит буквы строки в верхний регистр

В следующем примере применяются методы split() и join() для разбиения строки в список (по разделителям) и обратное объединение списка строк в строку:

```
>>> s = "This is an example."
>>> 1st = s.split(" ")
>>> print 1st
['This', 'is', 'an', 'example.']
>>> s2 = "\n".join(lst)
>>> print s2
This
is
an
example.
```

Для проверки того, оканчивается ли строка на определенное сочетание букв, применяется метод endswith():

```
>>> filenames = ["file.txt", "image.jpg", "str.txt"]
>>> for fn in filenames:
        if fn.lower().endswith(".txt"): print fn
file.txt
str.txt
```

Поиск в строке можно осуществить с помощью метода find(). Следующая программа выводит все функции, определенные в модуле оператором def:

```
import string
text = open(string.__file__[:-1]).read()
start = 0
while 1:
    found = text.find("def", start)
    if found == -1: break
    print text[found:found + 60].split("(")[0]
    start = found+1
```

Важным для преобразования текстовой информации является метод replace(), который рассматривается ниже:

```
>>>a="Это текст , в котором встречаются запятые , поставлен-
ные не так."
>>>b = a.replace(" ,", ",")
>>> print b
```

При работе с очень длинными строками или большим количеством строк применяемые операции могут по-разному влиять на быстродействие программы. Например, не рекомендуется многократно использовать операцию конкатенации для склеивания большого количества строк в одну. Лучше накапливать строки в списке, а затем с помощью joint() собирать в одну строку:

```
>>>a = " "
>>>for i in xrange(1000): a += str(i) # неэффективно!
>>>a=" ".join([str(i) for i in xrange(1000)])#эффект.
```

Если строка затем обрабатывается, можно применять итераторы, которые позволят свести использование памяти к минимуму.

22.8. Модуль StringIO

В некоторых случаях желательно работать со строкой как с файлом. Модуль StringIO как раз дает такую возможность. Открыть файл можно с помощью модуля StringIO(). При вызове без аргумента создается новый файл, при задании строки в качестве аргумента – файл открывается для чтения:

```
import StringIO
my_string = "1234567890"
f1 = StringIO.StringIO()
f2 = StringIO.StringIO(my_string)
```

Далее с файлами f1 и f2 можно работать как с обычными файловыми объектами. Для получения содержимого такого файла в виде строки применяется метод getvalue (): f1.getvalue().

Противоположный вариант (представление файла на диске в виде строки) можно реализовать на платформах Unix и Windows с использованием модуля mmap.

22.9. Модуль difflib

Для приблизительного сравнения двух строк в стандартной библиотеке предусмотрен модуль difflib.

Функция difflib.get_close_matches() выделяет n близких строк к заданной строке:

get_close_matches(word,possibilities,n=3,cutoff=0.6), где word – строка, к которой ищутся близкие строки. Здесь possibilities – список возможных вариантов, n – требуемое количество ближайших строк, cutoff – коэффициент (из диапазона [0,1]) необходимого уровня совпадения строк. Строки, которые при сравнении с word дают меньшее значение, игнорируются.

22.10. Регулярные выражения

Рассмотренных стандартных возможностей для работы с текстом достаточно не всегда. Например, в методах find() и replace() задается всего одна строка. В реальных задачах такая однозначность встречается довольно редко, чаще требуется найти или заменить строки, отвечающие некоторому шаблону.

Регулярные выражения (regular expressions) описывают множество строк с помощью специального языка, который будет рассмотрен ниже. (Строка, в которой задано регулярное выражение, будет называться шаблоном.)

Для работы с регулярными выражениями в Python используется модуль re. В следующем примере регулярное выражение помогает выделить из текста все числа:

```
>>> import re
>>> pattern = r"[0-9]+"
>>> number_re = re.compile(pattern)
>>> number_re.findall("122 234 65435")
['122', '234', '65435']
```

В этом примере шаблон pattern описывает множество строк, которые состоят из одного или более символов из набора "0", "1", ..., "9". Функция re.compile() компилирует шаблон в специальный Regex-объект, который имеет несколько методов, в том числе метод findall() для получения списка всех непересекающихся вхождений строк, удовлетворяющих шаблону, в заданную строку. То же самое можно было сделать и так:

```
>>> import re
>>> re.findall(r"[0-9]+", "122 234 65435")
['122', '234', '65435']
```

Предварительная компиляция шаблона предпочтительнее при его частом использовании, особенно внутри цикла. Следует заметить, что для задания шаблона использована необработанная строка. В данном примере она не требовалась, но в общем случае лучше записывать строковые литералы именно так, чтобы исключить влияние специальных последовательностей, записываемых через обратную косую черту.

22.11. Синтаксис регулярного выражения

Синтаксис регулярных выражений в Python почти такой же, как в других инструментах. Часть символов (в основном буквы и цифры) обозначают сами себя. Строка удовлетворяет (соответствует) шаблону, если она входит во множество строк, которые этот шаблон описывает. Различные операции используют шаблон по-разному. Так,

search() ищет первое вхождение строки, удовлетворяющей шаблону, в заданной строке, а match() требует, чтобы строка удовлетворяла шаблону с самого начала. Символы, имеющие специальное значение в записи регулярных выражений, приведены ниже.

Символ	Обозначение в регулярном выражении
"."	Любой символ
"^"	Начало строки
"$"	Конец строки
"*"	Повторение фрагмента нуль или более раз («жадное»)
"+"	Повторение фрагмента один или более раз («жадное»)
"?"	Предыдущий фрагмент либо присутствует, либо отсутствует
"{m, n}"	Повторение предыдущего фрагмента от m до n раз включительно («жадное»)
"[...]"	Любой символ из набора в скобках. Можно задавать диапазоны символов с идущими подряд кодами, например: a-z
"[^...]"	Любой символ не из набора в скобках
"\"	Обратная косая черта отменяет специальное значение следующего за ней символа
"I"	Фрагмент справа или фрагмент слева
"* ?"	Повторение фрагмента нуль или более раз («нежадное»)
"+ ?"	Повторение фрагмента один или более раз («нежадное»)
"{m, n} ?"	Повторение предыдущего фрагмента от m до n раз включительно («нежадное»)

Если A и B – регулярные выражения, то их конкатенация AB является новым регулярным выражением, причем конкатенация строк a и b будет удовлетворять AB, если a удовлетворяет A и b удовлетворяет B. Можно считать, что конкатенация – основной способ составления регулярных выражений.

Скобки, описанные ниже, применяются для задания приоритетов и выделения групп (фрагментов текста, которые потом можно получить по номеру или из словаря и даже сослаться в том же регулярном выражении).

Алгоритм, который сопоставляет строки с регулярным выражением, проверяет соответствие того или иного фрагмента строки регулярному выражению.

Например, строка "a" соответствует регулярному выражению "[a-z]",строка "fruit" соответствует "fruitIvegetable", а вот строка "apple" не соответствует шаблону "pineapple".

В таблице вместо *регвыр* может быть записано регулярное выражение, вместо *имя* – идентификатор, а флаги будут рассмотрены ниже.

Обозначение	Описание
1	2
" (регвыр)"	Обособляет регулярное выражение в скобках и выделяет группу
" (? •.регвыр)"	Обособляет регулярное выражение в скобках без выделения группы
" (?=регвыр)"	Просмотр вперед: строка должна соответствовать заданному регулярному выражению, но дальнейшее сопоставление с шаблоном начнется с того же места
" (? !регвыр) "	То же, но с отрицанием соответствия
" (?<=регвыр)"	Просмотр назад: строка должна соответствовать регулярному выражению. Не занимает места в строке, к которой применяется шаблон. Параметр регвыр должен быть фиксированной длины (то есть без "+" и "*")
" (? <! регвыр)"	То же, но с отрицанием соответствия
"(?p<имя>регвыр)"	Выделяет именованную группу с именем <имя>
" (?P=имя)"	Точно соответствует выделенной ранее именованной группе с именем <имя>
" (?#регвыр)"	Комментарий (игнорируется)
"(?(имя)рв1|рв2)"	Если группа с номером или именем <имя> оказалась определена, то результатом будет сопоставление с рв1, иначе – с рв2. Часть рв2 может отсутствовать

Окончание таблицы

1	2
" (? флаг)"	Задает флаг для всего данного регулярного выражения. Флаги необходимо задавать в начале шаблона

В таблице описаны специальные последовательности, в которых используется обратная косая черта.

Последовательность	Значение
1	2
"\1" – "\9"	Группа с указанным номером группы нумеруется начиная с 1
" \A"	Промежуток перед началом всей строки (почти аналогично "^")
" \Z"	Промежуток перед концом всей строки (почти аналогично " $")
" \b"	Промежуток между символами перед словом или после него
" \B"	Наоборот, не соответствует промежутку между символами на границе слова
" \d"	Цифра (аналогично "[0 – 9]")
" \D"	Не цифра (аналогично "[" 0 – 9]")
" \s"	Любой пробельный символ (аналогично "[\t\n\r\f\v]")
" \S"	Любой непробельный символ (аналогично "[^t\n\r\f\v]")
"\w"	Любая цифра или буква (зависит от флага LOCALE)
"\W"	Любой символ, не являющийся цифрой или буквой (зависит от флага LOCALE)

Флаги, используемые с регулярными выражениями:

Обозначение	Описание
"(?i)", re.I, re.IGNORECASE	Сопоставление проводится без учета регистра букв
"(?L)", re.L, re.LOCALE	Влияет на определение буквы в "\w", "\w", "\b", "\B" в зависимости от текущей культурной среды (locale)
"(?m)", re.M, re.MULTILINE	Если этот флаг задан, то "^" и " $" соответствуют началу и концу любой строки
"(?s)", re.S, re.DOTALL	Если задан, то "." соответствует также и символу конца строки " \n"
"(?x)", re.X, re.VERBOSE	Если задан, то пробельные символы, не экранированные в шаблоне обратной косой чертой, являются незначащими, а все, что расположено после символа "#", – комментариями. Позволяет записывать регулярное выражение в несколько строк для улучшения его читаемости и записи комментариев
"(?u)", re.U, re.UNICODE	В шаблоне и в строке использован Unicode

22.12. Методы объекта-шаблона

В результате успешной компиляции шаблона функцией re.compile() получается объект-шаблон типа SRE_Pattern, имеющий несколько методов, некоторые из них будут рассмотрены ниже:

- **match(s) –** сопоставляет строку s с шаблоном, возвращая в случае удачного сопоставления объект с результатом сравнения (объект SRE_Match), а в случае неудачного – None. Сопоставление начинается от начала строки;
- **search(s) –** аналогичен match(s), но ищет подходящую подстроку по всей строке s;
- **split(s[,maxsplit=0]) –** разбивает строку на подстроки, разделенные подстроками, заданными шаблоном. Если в шаблоне выделены группы, они попадут в результирующий список, перемежаясь с подстроками между разделителями. Если указан maxsplit, то будет произведено не более maxsplit разбиений;

- **findall(s) –** ищет все неперекрывающиеся подстроки s, удовлетворяющие шаблону;
- **finditer(s) –** возвращает итератор по объектам с результатами сравнения для всех неперекрывающихся подстрок, удовлетворяющих шаблону.

23. ГРАФИЧЕСКИЙ ИНТЕРФЕЙС

Пакет Tkinter, по сути, является оберткой для Tcl/Tk – известного графического пакета для сценарного языка Tcl. На примере этого пакета будем изучать основные принципы построения графического интерфейса пользователя. Почти все современные графические интерфейсы общего назначения строятся по модели WIMP – Window, Icon, Menu, Pointer (окно, иконка, меню, указатель). Внутри окон рисуются элементы графического интерфейса, которые для краткости будут называться виджетами (widget – штучка). Меню могут располагаться в различных частях окна, но их поведение достаточно однотипно: они служат для выбора действия из набора предопределенных действий. Пользователь графического интерфейса «объясняет» компьютерной программе требуемые действия с помощью указателя. Обычно указателем служит курсор мыши или джойстика, однако есть и другие «указательные» устройства. С помощью иконок графический интерфейс приобретает независимость от языка и в некоторых случаях позволяет быстрее ориентироваться в интерфейсе. Задачей графического интерфейса является упрощение коммуникации между пользователем и компьютером. Применение имеющихся в наличии у программиста (или дизайнера) средств при создании графического интерфейса нужно свести к минимуму, выбирая наиболее удобные пользователю виджеты в каждом конкретном случае. Кроме того, полезно следовать принципу наименьшего «удивления»: из формы интерфейса должно быть понятно его поведение. Для многих приложений такие действия выделены в отдельные серии экранов, называемые «мастерами» (wizards). Однако если приложение – скорее конструктор, из которого пользователь может строить нужные ему решения, то типичным действием является именно построение решения. Определить типичные действия нелегко, поэтому компромиссом может быть гибрид, в котором есть «мастера» и хорошие возможности для собственных построений. Тем не менее графический интерфейс не является самым эффективным интерфейсом во всех случаях.

Для многих предметных областей решение проще выразить с помощью деклараций на некотором формальном языке или алгоритма на сценарном языке.

23.1. Основы Tk

Основная черта любой программы с графическим интерфейсом – интерактивность. Программа не просто что-то считает (в пакетном режиме) от начала своего запуска до конца, ее действия зависят от вмешательства пользователя. Фактически графическое приложение выполняет бесконечный цикл обработки событий. Программа, реализующая графический интерфейс, событийно-ориентирована. Она ждет от интерфейса событий, которые обрабатывает сообразно своему внутреннему состоянию.

Эти события возникают в элементах графического интерфейса (виджетах) и обрабатываются прикрепленными к этим виджетам обработчиками. Сами виджеты имеют многочисленные свойства (цвет, размер, расположение), выстраиваются в иерархию принадлежности (один виджет может быть хозяином другого), имеют методы для доступа к своему состоянию.

Расположением виджетов (внутри других виджетов) ведают так называемые менеджеры расположения. Виджет устанавливается на место по правилам менеджера расположения. Эти правила могут определять не только координаты виджета, но и его размеры. В Tk имеются три типа менеджеров расположения: простой упаковщик (pack), сетка (grid) и произвольное расположение (place).

Но этого для работы графической программы недостаточно. Некоторые виджеты в графической программе должны быть взаимосвязаны определенным образом. Например, полоска прокрутки может быть взаимосвязана с текстовым виджетом: при использовании полоски текст в виджете должен двигаться, и наоборот, при перемещении по тексту полоска должна показывать текущее положение. Для связи между виджетами в Tk используются переменные, через которые виджеты и передают друг другу параметры.

23.2. Классы виджетов

Для построения графического интерфейса в библиотеке Tk отобраны следующие классы виджетов:

Button(Кнопка) – простая кнопка для вызова некоторых действий (выполнения определенной команды);

Canvas(Рисунок) – основа для вывода графических примитивов;

Checkbutton(Флажок) – кнопка, которая умеет переключаться между двумя состояниями при нажатии на нее;

Entry(Поле ввода) – горизонтальное поле, в которое можно ввести строку текста;

Frame(Рамка) – виджет, который содержит другие визуальные компоненты;

Label(Надпись) – виджет может показывать текст или графическое изображение;

Listbox(Список) – прямоугольная рамка со списком, из которого пользователь может выделить один или несколько элементов;

Menu(Меню) – элемент, с помощью которого можно создавать всплывающие (popup) и ниспадающие (pulldown) меню;

Menubutton(Кнопка-меню) – кнопка с ниспадающим меню;

Message(Сообщение) – аналогично *надписи*, но позволяет заворачивать длинные строки и менять размер по требованию менеджера расположения;

Radiobutton(Селекторная кнопка) – кнопка для представления одного из альтернативных значений. Такие кнопки, как правило, действуют в группе. При нажатии на одну из них кнопка группы, выбранная ранее, «отскакивает»;

Scale(Шкала) – служит для задания числового значения путем перемещения движка в определенном диапазоне;

Scrollbar(Полоса прокрутки) – полоса прокрутки служит для отображения величины прокрутки в других виджетах. Может быть как вертикальной, так и горизонтальной;

Text(Форматированный текст) – этот прямоугольный виджет позволяет редактировать и форматировать текст с использованием различных стилей, внедрять в текст рисунки и даже окна

Toplevel(Окно верхнего уровня) – показывается как отдельное окно и содержит внутри другие виджеты.

Все эти классы не имеют отношений наследования друг с другом – они равноправны. Этот набор достаточен для построения интерфейса в большинстве случаев.

23.3. События

В системе современного графического интерфейса имеется возможность отслеживать различные события, связанные с клавиатурой и мышью и происходящие на «территории» того или иного виджета. В Tk события описываются в виде текстовой строки – шаблона события, состоящего из трех элементов (модификаторы, тип события и детализация события).

Тип события	Содержание события
1	2
Activate	Активизация окна
ButtonPress	Нажатие кнопки мыши
ButtonRelease	Отжатие кнопки мыши
Deactivate	Деактивация окна
Destroy	Закрытие окна
Enter	Вхождение курсора в пределы виджета
Focusin	Получение фокуса окном
FocusOut	Потеря фокуса окном
Keypress	Нажатие клавиши на клавиатуре
KeyRelease	Отжатие клавиши на клавиатуре
Leave	Выход курсора за пределы виджета
Motion	Движение мыши в пределах виджета
MouseWheel	Прокрутка колесика мыши
Reparent	Изменение родителя окна
Visibility	Изменение видимости окна

Примеры описаний событий строками и некоторые названия клавиш приведены ниже:

- "<ButtonPress>" или просто "<3>" – щелчок правой кнопкой мыши (то есть третьей, если считать на трехкнопочной мыши слева направо);
- "<Shift-Double-Button-l>" – двойной щелчок мышью (левой кнопкой) с нажатой кнопкой <Shift>. В качестве модификаторов могут быть использованы следующие (список неполный): Control, Shift, Lock, Button1-Button или B1-B5, Meta, Alt, Double, Triple.

Просто символ обозначает событие – нажатие клавиши. Например, "k" – то же, что "<KeyPress-k>". Для неалфавитно-цифровых клавиш есть специальные названия: <Cancel>, <Backspace>, <Tab>, <Return>, <Shift_L>, <Control_L>, <Alt_L>, <Pause>, <Caps_Lock>, <Escape>, Prior>, <Next>, <End>, <Home>, <Left>, <Up>, <Right>, <Down>, <Prin>t, <Insert>, <Delete>, <Fl>, <F2>, <F3>, <F4>, <F5>, <F6>, <F7>, <F8>, <F9>, <F10>, <F11>, <F12>, <Num_Lock>, <Scroll_Lock>, <space, less>.

Здесь <space> обозначает пробел, а <less> – знак меньше. <Left>, <Right>, <Up>, <Down> – стрелки. <Prior>, <Next> – это <PageUp> и <PageDown>. Остальные клавиши более или менее соответствуют надписям на стандартной клавиатуре. Следует заметить, что <Shift_L> в отличие от Shift нельзя использовать как модификатор.

В конкретной среде комбинации, означающие что-то особенное в системе, могут не дойти до графического приложения. Например, известный всем <Ctrl-Alt-Del>. Следующая программа позволяет печатать направляемые виджету события, в частности KeyPress, а также анализировать, как различные клавиши можно представить в шаблоне события:

```
from Tkinter import *
tk = Tk()# основное окно приложения
txt = Text(tk)# виджет, принадлежащий окну tk
txt.pack()#ф.обраб. событ.располаг. менеджером pack
def event_info(ev):
    txt.delete("1.0",END) #удаляется с нач. до конца
    for k in dir(ev): #цикл по атрибутам события
        if k[0] != "_":  #только неслужебные атрибуты
       # готовится описание атрибута события
```

```
            ev="%s:%s\n" % (k,repr(getattr(ev,k)))
            txt.insert(END, ev)
    txt.bind("<KeyPress>",' event_info)
    tk.mainloop()    # главный цикл обработки событий
```

При нажатии клавиши Esc в окне можно увидеть примерно следующее:

- char: '\x1b' delta: 9 height: 0 keycode: 9
- keysym: 'Escape' keysym_num: 65307
- num: 9 send_event: False serial: 159 state: 0 time: -1072960858 type: '2'
- widget: <Tkinter.Text instance at Ox401e268c> width: 0
- x: 83 x_root: 448
- y: 44 y_root: 306

Некоторые атрибуты:

- char – нажатый символ (для некоторых событий – ??);
- height, width – высота и ширина;
- Focus – был ли в момент события фокус у окна;
- Keycode – код символа (скан-код клавиатуры);
- Keysym – символическое имя клавиши;
- Serial – серийный номер события, увеличивается по мере возникновения событий;
- Time – время возникновения события;
- widget – виджет, в котором возникло событие;
- x,y – координаты указателя в виджете во время события;
- x_root, y_root – координаты указателя на экране во время события.

Совсем необязательно, чтобы события обрабатывал виджет, который их первично принял. Например, можно перенаправить все события внутри подчиненных виджетов на данный виджет с помощью метода grab_set(), (grab_reiease() освобождает виджет от этой обязанности). В Tk существуют и другие возможности управления событиями, которые можно изучить по документации.

23.4. Создание и конфигурирование виджета

Создают виджет путем вызова конструктора соответствующего класса. Вызов конструктора имеет следующий синтаксис:

Widget([master[,option=value, …]]).

Здесь Widget – класс виджета, master – виджет-хозяин, option и value – конфигурационная опция и ее значение (таких пар может быть несколько).

Каждый виджет имеет свойства, которые можно устанавливать (конфигурировать) с помощью методов config() (или configure()) и читать с помощью методов, подобных методам работы со словарями. Ниже приведен возможный синтаксис для работы со свойствами:

```
widget.config(option=value, ...)
widget["option"] = value
value = widget["option"]
widget.keys()
```

В случае, когда имя свойства совпадает с ключевым словом языка Python, принято использовать после имени одиночное подчеркивание. Так, свойство class нужно задавать как class_, а to как to_.

Изменять конфигурацию виджета можно в любой момент. Это изменение прорисуется на экране по возвращении в цикл обработки событий или при явном вызове update_idletasks().

Пример, приведенный ниже, показывает окно с двумя виджетами внутри – полем ввода и надписью. С помощью переменной надпись напрямую связана с полем ввода. В этом примере нарочно используется очень много свойств, чтобы продемонстрировать возможности по конфигурированию:

```
from Tkinter import *
tk = Tk()
tv = StringVar()
Label(tk,textvariable=tv, relief="groove",
  borderwidth=3,
  font=("Courier",20,"bold"), justify=LEFT,
  width=50,
  padx=10,
  pady=20,
  takefocus=False,).pack()
Entry(tk, textvariable=tv, takefocus=True).pack()
tv.set("123")
tk.mainloop()
```

Виджеты конфигурируются прямо при создании. Более того, виджеты не связываются с именами, их только располагают внутри виджета-окна. Свойства виджета:

- textvariable (текстовая переменная);
- relief (рельеф);
- borderwidth (ширина границы);
- justify (выравнивание);
- width (ширина, измеряемая в знакоместах);
- padx и pady (прослойка в пикселях между содержимым и границами виджета);
- takefocus (возможность принять фокус при нажатии клавиши Tab);
- font (шрифт, один из способов его задания).

Эти свойства достаточно типичны для многих виджетов, хотя иногда единицы измерения могут отличаться. Например, для виджета Canvas ширина задается в пикселях, а не в знакоместах.

В следующем примере демонстрируются возможности по назначению цветов фону, переднему плану (тексту), выделению виджета (подсветка границы) в активном состоянии и при отсутствии фокуса:

```
from Tkinter import *
tk = Tk()
tv =StringVar()
Entry(tk,textvariable=tv,takefocus=True,
  borderwidth=10).pack()
mycolor1 = "#%02X%02X%02X" % (200, 200, 20)
Entry(tk,textvariable=tv,takefocus=True,
  borderwidth=10,
  foreground=mycolorl,        # fg, текст виджета
  background="#0000FF",       # bg, фон виджета
  highlightcolor='green',     # подсветка при фокусе
  highlightbackground='red', # подсветка без фокуса
  ).pack()
tv.set("123")
tk.mainloop()
```

При желании можно задать стилевые опции для всех виджетов сразу с помощью метода tk_setPalette(). Помимо использованных выше свойств, с помощью данного метода можно включить selectForeground() и selectBackground (передний план и фон выделения), selectColor (цвет в выбранном состоянии, например у Checkbutton()), insertBackground (цвет точки вставки) и некоторые другие.

Пример оформления графика в пакете Tkinter:

```
# -*- coding: utf-8 -*-
from Tkinter import *
from bsc_groups import *
from math import*
##======Function
def f(x): return x**2*sin(x)
##======Calculating
bl=Bl('x','y')
x=-0.5
xk=2.0*pi
dx=0.01
while x<=xk:
    y=f(x)
    bl.add(x=x,y=y)
    x+=dx
##======Labels
crl=["red","black","#0DF","#C06","#A64","#AC3"]
bg='White'
Lbd =[Bd(200,55,'Функция f=x**2*sin(x)',crl[0],bg)]
tf= "y=[%6.3f  %6.3f]"
xmin= min(bl.L['y'])
xmax= max(bl.L['y'])
Lbd+=[Bd(165,600,tf % (xmin, xmax),crl[0],bg)]
xmin= min(bl.L['x'])
xmax= max(bl.L['x'])
Lbd+=[Bd(165, 615,tf % (xmin, xmax),crl[1],bg)]
##======Tkinter
A,B=500,700
tk=Tk()
```

```
tk.title('График функции')
fr=Frame(tk)
fr.pack()
c=Canvas(fr,bg=bg,width=A,height=B)
c.pack(expand=1,fill=BOTH)
#=====Output
Lx= bl.L['x']
Ly= bl.L['y']
desine_l(60,130,400,400,bg,crl,Lx,[Ly],0,50,50,c)
for ibd in Lbd: ibd.show(c)
tk.mainloop()
```

24. ИЕРАРХИЯ СТАНДАРТНЫХ ИСКЛЮЧЕНИЙ

Язык Python поддерживает много встроенных исключений. Все эти исключения входят в состав модуля exceptions, который всегда загружается автоматически перед выполнением любой программы. Все исключения происходят от базового класса exception, исключение importerror является подклассом класса standarderror, который, в свою очередь, является подклассом класса exception.

Exception. Базовый класс. Все классы исключений являются его подклассами. От этого класса должны также происходить все классы исключений, определяемых пользователем.

SystemExit. Исключение в чистом виде, поскольку в действительности оно не связано с появлением ошибок. Оно применяется для выхода из программы. Важно отметить, что это исключение не возвращает сообщений обратной трассировки.

StandardError. Базовый класс для всех ошибок (безусловно, кроме SystemExit).

KeyboardInterrupt. Исключение активизируется при нажатии клавиши прерывания, например при использовании комбинации клавиш <CTRL+C>.

ImportError. Активизируется, если интерпретатор Pythonне может найти импортируемый модуль.

EnvironmentError. Базовый класс для ошибок, возникающих вне среды Python. Классы **IOError** и **OSError** являются его подклассами.

IOError. Исключение активизируется в результате возникновения ошибок при выполнении операций ввода/вывода.

OSError. Исключение активизируется при получении сообщений об ошибках от операционной системы, обычно связанных с применением модуля os.

EOFError. Исключение активизируется при получении сообщения о достижении конца файла (EOF – End-of-File).

RuntimeError. Исключение особого типа, которое активизируется в результате ошибок, не относящихся ни к одному из прочих исключений.

NotImplementedError. Исключение активизируется при попытках вызова методов или функции, не реализованных в интерпретаторе.

```
>>> def updateregistry():
>>> raise NotImplementedError
```

NameError. Исключение активизируется, если интерпретатор обнаруживает в программе имя, не принадлежащее ни к локальному, ни к глобальному пространству имен.

UnboundLocalError. Новое исключение, которое было разработано для версии 1.6. Оно является подклассом исключения NameError и вызывает ошибку, если не определена локальная переменная, применяемая в программе.

AttributeError. Исключение активизируется при ошибках, связанных со ссылкой на атрибут и присваиванием атрибута. Обратите внимание, что, начиная с версии 1.6, это исключение имеет более дружественное сообщение об ошибке, которое может нарушить работу кода, разработанного в предположении, что сообщение будет эквивалентно имени атрибута.

SyntaxError. Активизируется при синтаксических ошибках.

TypeError. Исключение активизируется при попытке применить функциональную операцию к объекту, тип которого не соответствует этой операции.

AssertionError. Исключение активизируется оператором **assert,** если результат вычисления указанного в нем выражения является ложным.

LookupError. Базовый класс ошибок, связанных доступом по индексу и ключу. Его подклассами являются классы IndexError и KeyError.

IndexError. Исключение активизируется при возникновении ошибок, связанных с выходом индекса за пределы последовательности.

KeyError. Исключение активизируется, если ключ не найден в словаре.

ArithmeticError. Базовый класс для арифметических ошибок. Его подклассами являются классы OverflowError, ZeroDivisionError и FloatingPointError.

OverflowError. Исключение активизируется, если полученный результат настолько велик, что приводит к арифметическому переполнению.

ZeroDivisionError. Исключение активизируется при попытке деления на нуль.

FloatingPointError. Исключение активизируется при ошибках выполнения операций с плавающей точкой. Обратите внимание, что в системе Linux для использования данного исключения необходимо разрешить обработку сигнала SIGFPE с применением модуля fpectl.

ValueError. Исключение активизируется при попытке выполнения операции над переменной с правильным типом, но с неправильным значением.

SystemError. Исключение активизируется при возникновении внутренней ошибки интерпретатора Python.

MemoryError. Исключение активизируется при возникновении неисправимой ошибки, связанной с нехваткой памяти. Поскольку классы исключений являются подклассами других классов исключений (называемых базовыми классами), то существует возможность перехватывать ошибки/исключения сразу нескольких типов с использованием одной фразы except. Исключения базовых классов никогда не активизируются, но могут применяться для перехвата ошибок.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Бизли Д. М. Язык программирования Python : справочник : пер. с англ. / Д. М. Бизли. – Киев : ДиаСофт, 2000.
2. Гифт Н. Python в системном администрировании UNIX и Linux : пер. с англ. / Н. Гифт, Д. Джонс. – СПб. : Символ-Плюс, 2009.
3. Лейнингем И. Освой самостоятельно Python за 24 часа : пер. с англ. / И. Лейнингем. – М. : Издательский дом «Вильямс», 2001.
4. Лесса А. Python. Руководство разработчика : пер. с англ. / А. Лесса. – СПб. : ДиасофтЮП, 2001.
5. Лутц М. Изучаем Python : пер. с англ. / М. Лутц. – СПб. : Символ-Плюс, 2009.
6. Лутц М. Программирование на Python : пер.с англ. / М. Лутц. – СПб. : Символ-Плюс, 2002.
7. Саммерфельд М. Программирование на Python 3. Подробное руководство : пер. с англ. / М. Саммерфельд. – СПб. : Символ-Плюс, 2009.
8. Сузи Р. А. Python / Р. А. Сузи. – СПб. : БХВ-Петербург, 2002.
9. Сузи Р. А. Язык Python и его применения : учеб. пособие / Р.А. Сузи. – М. : Интернет-Университет информационных технологий: БИНОМ. Лаборатория знаний, 2006.
10. Язык программирования Python / Г. Россум [и др.]. – СПб. : АНО «Институт логики» – Невский диалект, 2001.

Учебное издание

Буйначев Сергей Константинович,
Боклаг Наталья Юрьевна

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ PYTHON

Редактор *И. В. Коршунова*
Компьютерная верстка *Е. В. Суховой*

Подписано в печать 27.05.2014. Формат 60×90 1/16.
Бумага типографская. Плоская печать. Усл. печ. л. 5,75.
Уч.-изд. л. 5,0 Тираж 100 экз. Заказ № 1159.

Издательство Уральского университета
Редакционно-издательский отдел ИПЦ УрФУ
620049, Екатеринбург, ул. С. Ковалевской, 5
Тел.: 8 (343) 375-48-25, 375-46-85, 374-19-41
E-mail: rio@urfu.ru

Отпечатано в Издательско-полиграфическом центре УрФУ
620075, Екатеринбург, ул. Тургенева, 4
Тел.: 8 (343) 350-56-64, 350-90-13
Факс: 8 (343) 358-93-06
E-mail: press-urfu@mail.ru

Для заметок