

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ
ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ТУЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

Институт прикладной математики и компьютерных наук
Кафедра информационной безопасности

***МЕТОДИЧЕСКИЕ УКАЗАНИЯ К
ЛАБОРАТОРНЫМ РАБОТАМ***
по дисциплине

ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ

Уровень профессионального образования: *высшее образование –
бакалавриат*

Направление подготовки: *090303 Прикладная информатика*

Профиль подготовки: *Прикладная информатика в экономике*

Квалификация выпускника: *бакалавр*

Структурное программирование на языке программирования Python. Часть 1.

1. ЦЕЛЬ И ЗАДАЧИ РАБОТЫ: Изучение основ структурного программирования на языке программирования Python.

2. ОБЩИЕ ПОЛОЖЕНИЯ

Начинать изучать язык программирования, неважно какой, необходимо с самых его основ, элементарных базовых операций и конструкций., изучить его работу с памятью. Анализировать и сравнивать реализации идентичных операций с уже известным вам языком. В данном случае Python часто будет сравниваться с Pascal.

По синтаксису они очень схожи, по крайней мере, гораздо больше чем с C++. Семантика этих языков чуть ли не одинаковая, а синтаксис большинства операторов различается незначительно. Python, кстати, даже немного более гибок в некоторых случаях, нежели Pascal. Например, функции могут возвращать не одно значение как в Pascal, а несколько. Наиважнейшее отличие этих языков кроется в их архитектуре, в способах хранения и обработки данных.

1.1. Операции

Как и в любом другом языке, операции поделены на математические и логические. Синтаксис записи команд прямой (не как в VisualLISP), все очень доступно и понятно. Все как было в Pascal.

Математические		Логические	
Оператор	Описание	Оператор	Описание
+	Сложение	<=	Меньше или равно
-	Вычитание	>=	Больше или равно
*	Умножение	==	Равно
/	Деление	!=	Не равно
%	Остаток от деления	and	И
**	Возведение в степень	or	Или
=	Присвоение	not	Инверсия

Отличаются 4 оператора: «присвоение», «равно», «не равно» и «возведение в степень». Вернее сказать «отличаются 3 оператора и добавлен новый», так как оператора возведения числа в степень в Pascal нет.

Python	Pascal	Семантика
b = a	b := a	[b присвоить a]
b == a	b = a	[b равно a]
b != a	b <> a	[b не равно a]
b = a ** 3	b := exp (a * ln (3))	[b присвоить a ³]

Остальное одинаково, логические операции возвращают булево значение – True или False.

Функции конвертирования типов	
float(a)	в вещественное
int(a)	в целое (отбрасывает дробную часть)
str(a)	в строку

1.2. Вывод информации

Вывод на экран сообщений, значений переменных, любую информацию осуществляет оператор “print” (также в интерактивном режиме можно вывести результат какого-либо действия или выполнить выражение и без оператора print). Если надо отобразить сразу несколько значений, то они перечисляются через запятую и выводятся через пробел.

```
>>> 9+5-4
10
>>> print 9+5-4
10
>>> 6,8,4
(6, 8, 4)
>>> print 6,8,4
6 8 4
```

Как видно из примера одновременный вывод нескольких чисел 6, 8, 4 без оператора получается в виде списка. Это происходит из-за того, что когда Python выводит значение выражения в командном режиме, он использует тот же формат, что используется при вводе этого значения. Например, при выводе строк он заключает их в кавычки. Команда print также выводит значение выражения, но в случае со строками, она выводит содержимое строки без кавычек.

В скриптах, выражения вообще ничего не выводят, если в них нет инструкции print.

Например, следующий скрипт не выводит ни одной строки:

```
a=17
3.2
'Смотрю "Малахов+" и пишу диплом'
a=a+1
```

Сейчас было продемонстрировано, как можно выводить кавычки – заключить строку не в кавычки, а в апострофы, а кавычки использовать внутри.

Можно решить проблему вывода кавычек и других вопросов и по-другому – использовать экранирующие символы.

Уже прекрасно известно, что в кодовых таблицах (наборах цифровых кодов, обозначающих различные символы, которые компьютер может выводить на экран) есть целая группа так называемых непечатаемых символов. Непечатаемые символы используются для управления вводом/выводом. Самые часто используемые из них: знак табуляции, перенос на новую строку и знак «возврата каретки». Т.к. в кодовой таблице

нет символов, отображаемых на экране, для их обозначения придумали специальные последовательности:

Последовательность	Назначение
\t	Табуляция
\n	Перевод на новую строку
\r	Возврат «каретки» (курсора) в начало строки

Таким образом, если в программе нужно вывести текст из нескольких строк, табулированный текст, то на помощь приходят специальные последовательности:

```
>>> print "Горox\rП\n\tсыпуч"
Горox
    Сыпуч
```

Как выводится такая строка "Горох\rП\n\tсыпуч"? Сначала выводится строка "Горох" – только после нее интерпретатор встречает первую специальную последовательность: "\r" – символ возврата «каретки» на начало строки. Затем выводится символ "П", но происходит это на позиции первого символа. При этом уже имеющийся на этой позиции символ заменяется новым. После этого последовательности "\n" и "\t" дают указание произвести перевод на новую строку и оставить отступ в одну табуляцию. Далее выводится оставшаяся часть строки "сыпуч".

Разумеется, не стоит применять специальные последовательности там, где это не обязательно, иначе программу будет сложно и неудобно читать.

Теперь рассмотрим такую ситуацию: предположим, что необходимо вывести, например, такую строку "Main \template/".

```
>>> print "Main \template/"
Main  emplate/
```

Интерпретатор распознал последовательность "\t", и вставил специальный символ — табулировал текст, а это совсем не то, чего мы хотели добиться.

В подобных ситуациях перед специальным символом "\" ставится так называемый экранирующий символ: еще один обратный слэш "\".

```
>>> print "Main \\template/"
Main \template/
```

Кавычки вывести тоже можно при помощи экранирования:

```
>>> print "Смотрю \"Малахов+\" и пишу диплом"
Смотрю "Малахов+" и пишу диплом
```

В тексте программ тоже используются специальные символы. Более того, символ переноса на новую строку является признаком завершения выражения, или, иначе говоря, разделителем выражений. Интересно, что в Python его тоже можно экранировать. Зачем? Ну, например, если выражение слишком длинное, и для удобства чтения программы предпочтительно его разбить на несколько строк:

```
>>> print "\"Я не верю в чудеса, но летит\" \n\t\t\t\t\tА.Рыбак"
"Я не верю в чудеса, но летит машина в небеса!"
                                     А. Рыбак
```

В конце первой строки стоит символ "\" (обратите внимание: он находится за пределами строкового значения), который экранирует символ переноса строки. Таким образом разделитель выражений Питона игнорируется интерпретатором, и выражение благополучно продолжается на следующей строке.

1.3. Переменные

Прежде чем говорить о типах данных переменных, нужно узнать как вообще хранятся эти переменные в памяти компьютера и что это такое “динамическая типизация”?

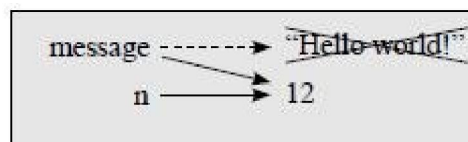
Реализация переменных внутри Python несколько иная, нежели в другом языке, подобном Pascal. Если в языках C++ или Pascal переменная – это имя ячейки памяти, в которой хранится значение, то в Python переменная – это ссылка на ячейку памяти. Это и объясняет иной подход к организации хранения объектов в памяти.

Для того, чтобы «запомнить» значение достаточно присвоить (сопоставить) его переменной. Объявлять переменные как в Pascal тоже не нужно, более того, интересно, что одной и той же переменной можно присваивать разные типы данных в любой момент времени. Плюс этого – экономное распределение памяти, сокращение кода. Минус – возможная путаница, требует больше внимания. В этом смысле Pascal более дисциплинированный.

Тип переменной синхронизируется с типом присваиваемого ей значения. Рассмотрим пример:

```
>>> message = "Hello world!"
>>> n = 12
>>> type(message)
<type 'str'>
>>> message = n
>>> print message
12
>>> type(message)
<type 'int'>
```

Этот пример интересен по нескольким причинам. Во-первых, в нем явно видно использование возможности присваивать значение одной переменной другой, изначально разнотипных: message – текст, n – целое число. Конструкция message = n работает аналогично присваиванию переменной значения: переменной message присваивается значение переменной n. При этом значение 12 хранится в памяти только один раз, а старое значение переменной message “Hello world!” стирается – Python довольно экономно расходует память.



Во-вторых, как видно из примера, переменная message после присваивания ей значения n поменяла свой тип. Далеко не каждый язык программирования «умеет» это делать так просто.

Функция type() возвращает тип введенного ей параметра, причем этот параметр необязательно должен быть переменной, скажем, может быть и функцией.

```
>>> type(message)           # тип message
<type 'int'>
>>> type("vkontakte.ru")    # тип "vkontakte.ru"
<type 'str'>
>>> type(1==2)               # тип результата 1==2
<type 'bool'>
>>> type(type)               # тип функции type
<type 'type'>
```

- так обозначается комментарий в Python

```
|   int – целочисленный тип      (Pascal: integer)
|   str – строковый тип         (Pascal: string)
|   bool – логический тип       (Pascal: boolean)
|   type – тип-типа-тип
```

Имена переменных могут быть произвольной длины. При составлении имен переменных в Python можно использовать любые латинские буквы, цифры и знак подчеркивания (_). Названия переменных не могут начинаться с цифры. Естественно и зарезервированные слова не могут быть именем переменной. Вот список зарезервированных слов в Python версии 2.5, их всего 31:

and	as	assert	break
class	continue	def	del
elif	else	except	exec
finally	for	from	global
if	import	in	is
lambda	not	or	pass
print	raise	return	try
while	with	yield	

Список ключевых слов можно узнать так:

```
>>> import keyword
>>> keyword.kwlist
```

Python чувствителен к регистру в именах переменных, функций. То есть, переменные A и a – это ссылки на физически разные адреса памяти.

Убрать связь имени с объектом можно с помощью оператора del. В этом случае, если объект не имеет других ссылок на него, он будет удален. Для управления памятью Python использует менеджер подсчета ссылок (reference counting), для удаления наборов объектов с зацикленными ссылками - сборка мусора (garbage collection).

1.4. Ввод данных с клавиатуры

Чтобы пользователь мог задать значения переменным с клавиатуры, нужно воспользоваться специальными функциями. Для текстовых и числовых переменных они разные.

```
|   input(<подсказка>)         – ввод числовых значений
|   raw_input(<подсказка>)    – ввод строки
```

Подсказка необязательна, но дает понять пользователю, что нужно ввести. Рассмотрим пример использования:

```
>>> a = raw_input("Ваш вопрос: ")
Ваш вопрос: Сколько Вам лет?
>>> a = input(a)
Сколько Вам лет? 19.5
>>> print a
19.5
```

Здесь в переменную *a* сначала вносится значение типа текст «Сколько Вам лет? », а затем это значение используется как подсказка при следующем вводе. Причем ввод происходит в ту же переменную. Понятно, что подсказкой может быть значение переменной, и необязательно строкового типа.

2. Встроенные типы данных Python

Первое представление о переменных и типах данных у нас есть. Настало время углубиться и изучить все подробнее. Все данные в Python представлены объектами. Имена являются лишь ссылками на эти объекты и не несут нагрузки по декларации типа. Ниже представлена полная карта встроенных типов:

- специальные типы: **None**, **NotImplemented** и **Ellipsis**;
- числа;
 - целые
 - обычное целое **int**
 - целое произвольной точности **long**
 - логический **bool**
 - число с плавающей точкой **float**
 - комплексное число **complex**
- последовательности;
 - неизменяемые:
 - строка **str**;
 - unicode-строка **unicode**;
 - кортеж **tuple**;
 - изменяемые:
 - список **list**;
- отображения:
 - словарь **dict**
- объекты, которые можно вызвать:
 - функции (пользовательские и встроенные);
 - функции-генераторы;
 - методы (пользовательские и встроенные);
 - классы (новые и "классические");
 - экземпляры классов (если имеют метод **__call__**);
- модули;
- классы;
- экземпляры классов;
- файлы **file**;
- вспомогательные типы **buffer**, **slice**.

Некоторые типы рассмотрим подробнее.

2.1. Типы int и long

Два типа: int (целые числа) и long (целые произвольной точности) служат моделью для представления целых чисел. Первый соответствует типу integer в языке Pascal. Набор операций над числами - достаточно стандартный как по семантике, так и по обозначениям.

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

Стоит заметить, что если в результате операции получается значение, выходящее за рамки допустимого, тип int может быть неявно преобразован в long:

```
>>> type(-2147483648)
<type 'int'>
>>> type(-2147483649)
<type 'long'>
```

Еще одна особенность при арифметике с типом int:

```
>>> a = 6
>>> a / 4
1
```

Что за ошибка?! Почему ответ 1?!...

Дело в том, что результат арифметических действий Python автоматически приводит к типу наиболее сложного операнда. Здесь они оба int, то есть целочисленные, поэтому дробную часть Python просто откинул. В этом случае нужно делить не на 4, а на 4.0. Или же воспользоваться функциями конвертирования типов:

```
>>> float(a)/4      # Pascal: inttofloat
1.5
```

2.2. Тип float

Соответствует Pascal-типу real. Записывается вполне традиционным способом либо через точку, либо в нотации с экспонентой:

```
>>> pi = 3.1415926535897931
>>> pi ** 40
7.6912142205156999e+19
```

Кроме арифметических операций, можно использовать операции из модуля math. Из полезных встроенных функций можно вспомнить округление - round() и модуль - abs().

2.3. Тип bool (логический)

Подтип целочисленного типа для "канонического" обозначения логических величин. Два значения: True (истина) и False (ложь) – как и в случае с Pascal.

```
>>> 8==2+2*2
False
>>> 1!=5
True
```

Следует отметить, что Python даже не вычисляет второй операнд логических операций `or` и `and`, если ее исход ясен по первому операнду. Таким образом, если первый операнд истинен, то для операции `or` он и возвращается как результат, в противном случае возвращается второй операнд, если истинен он.

Для операции `and` транслятор не вычисляет истинность второго условия, если ложно первое.

2.4. Тип `str` и тип `unicode`

В Python строки бывают двух типов: обычные и Unicode-строки. Фактически строка - это последовательность символов. 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. Управляющие последовательности внутри строковых литералов задаются обратной косой чертой (`\`). Скрипт-пример написания строковых литералов:

```
# -*- coding: cp1251 -*-
s1 = "строка1"
s2 = 'строка2\nс переводом строки внутри управляющим символом'
s3 = """строка3
с переводом строки внутри тройными кавычками"""
u1 = u'\u043f\u0440\u0438\u0432\u0435\u0442' # слово привет в Unicode
u2 = u'Еще пример, Unicode-строка'
```

Для строк имеется еще одна разновидность: необработанные строковые литералы. Чтобы вывести в таком виде нужно перед заключающими строку кавычками поставить `r` (`raw`). В этих литералах обратная косая черта и следующие за ней символы не интерпретируются как спецсимволы, а вставляются в строку "как есть":

```
>>> sample = r"(\d)=\1\temp\nT"
>>> print sample
>>> (\d)=\1\temp\nT
```

Набор операций над строками включает конкатенацию `+`, повтор `*`, форматирование `%`. Также строки имеют большое количество методов, некоторые из которых приведены ниже. Полный набор методов (с аргументами) можно получить в справке по команде `help(str)`.

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

О других типах данных будет рассказано немного позже.

3. Стандартные управляющие конструкции

В этом разделе будут рассмотрены самые необходимые управляющие конструкции – киты, на которых стоят все алгоритмы. Э.Дейкстра как-то сказал: “Любой алгоритм

можно реализовать, используя лишь три управляющие конструкции: последовательное выполнение, ветвление и цикл”. И сумел доказать это.

Действительно, без этих конструкций не обойтись даже в самой примитивной программке. Все это уже изучено вдоль и поперек в Pascal – вложенные циклы в различных интерпретациях, многоуровневое ветвление. Схематическое представление конструкций (Рис. 2.1):

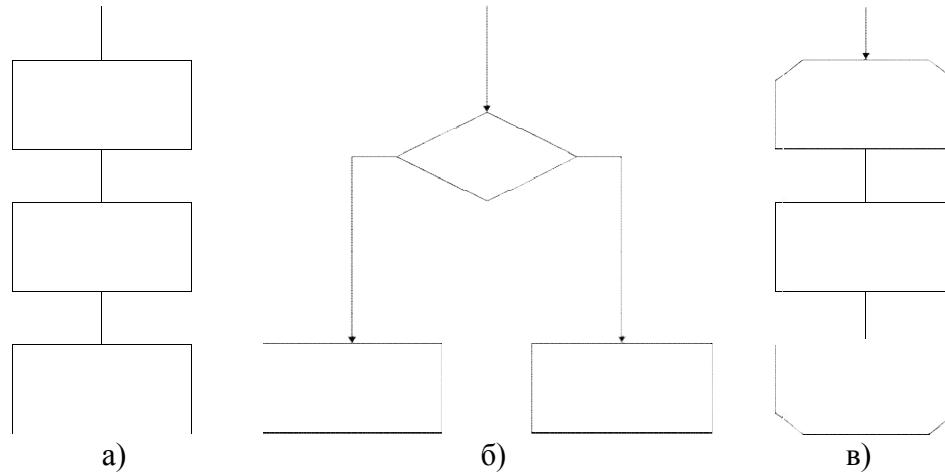


Рис. 2.1 - Основные управляющие конструкции:
а – последовательное выполнение; б – ветвление; в – цикл.

Посмотрим, как это реализовано в Python.

3.1. Ветвление, логический оператор

Вот мы и дошли до того места, где ярко выражается характерная черта языка Python – отступы, как прием структурирования программы. Если в Pascal совершенно неважно как располагать строки кода относительно друг друга (но грамотная структура программы – признак хорошего стиля программирования и существенно упрощает читаемость кода), то в этом языке это несет и синтаксический и семантический смыслы. Отступы – это аналог операторских скобок в Pascal. Python’исты призывают делать каждый отступ в 1 tab/4 пробела. Так что если нужно подтянуть дисциплинированность в написании кода – добро пожаловать в Python!

Синтаксис условного оператора в Python выглядит, как немного модифицированный Pascal’евский:

```
if [логическое условие 1]:  
    [последовательность выражений 1]  
elif [логическое условие 2]:  
    [последовательность выражений 2]  
elif [логическое условие 3]:  
    [последовательность выражений 3]  
...  
else:  
    [последовательность выражений n]
```

Поведение этой цепочки лучше всего смотреть на примере. Напишем скрипт и проанализируем его выполнение:

```
# -*- coding: cp1251 -*-
text = "PostCards"
if (2+3==4) and (2+2*2==8):           # оба False
    print "Это не выведет"
elif (1+1==11) or (text=="PostCards"): # True
    print "Вот это работает, т.к. "\
        "второе условие истинно"
elif (2==2):                           # True
    print "А сюда не пойдет, хоть "\
        "условие и выполняется"
else:
    print "Это тоже не выведет"
```

В результате выведется лишь строка:

```
>>>
Вот это работает, т.к. второе условие истинно
```

Первая строка – комментарий того, что использовалась кодовая страница 1251 для корректного отображения кириллицы. В первом условии обе проверки дают False. Помните, что интерпретатор даже не проверяет второе условие в `and`, если уже на первом возврат значения False? Далее следует аналог вложенного условия `else if` в Pascal. Одно условие истинно. Выполняется ветка. Здесь абсолютное значение имеют отступы, ведь выполнится все то, что стоит под условием с отступом в одну табуляцию. До следующей проверки дело не дойдет, т.к. интерпретатор проверяет до первого попадания в True выражения из условий. Поэтому ветка `else` выполняется в последнюю очередь, если все предшествующие “терпят крах”. Поэтому здесь она не выполнится.

Ветки `elif` и `else` необязательны. В общем все знакомо.

3.2. Циклы

В Python имеются два вида циклов: **цикл while** (общий оператор, выполняется некоторое действие) и **цикл for** (удобен для всех значений последовательностей).

Оператор цикла `while` имеет синтаксис:

```
while [условие]:
    [последовательность выражений]
else:
    [последовательность выражений]
```

Он расширен по сравнению с Pascal. Появилась альтернативная ветка. Она необязательна и выполняется лишь раз, в конце, в случае невыполнения условия цикла. Следующий пример демонстрирует повтор ввода данных пользователем, пока не будет получено верное значение:

```
# -*- coding: cp1251 -*-
import random
x=random.randint(0,9)
print "PC: загадано число [0..9]"
while not (x==input("Ваш выбор: ")):
    x=random.randint(0,9)
```

```

        print "Неверно! Попробуйте снова!"
    else:
        print "Правильно!"

```

Вначале мы подключили модуль random оператором import. Это тоже самое, что и Uses в Pascal. Модуль random содержит функции, позволяющие выбрать число псевдослучайным образом. В данном случае использована функция randint, генерирующая целое число в диапазоне [0..9], но есть и много других, например choice, выбирающая элемент последовательности, например строки.

```

>>> import random
>>> print random.choice("PostCards")
0

```

Конструкция else сокращает код, данный пример это показывает.

Для большей гибкости при организации циклов применяются операторы break (прервать) и continue (продолжить). Первый позволяет прервать цикл, а второй - продолжить цикл, перейдя к следующей итерации (если, конечно, выполняется условие цикла).

В Python цикл for выполняет тело цикла для каждого элемента последовательности. В следующем примере выводится таблица умножения:

```

for i in range(1, 10):
    for j in range(1, 10):
        print "%2i" % (i*j),
    print

```

Функция range() порождает список целых чисел из полуоткрытого диапазона [1..10). Перед каждой итерацией счетчик цикла получает очередное значение из этого списка. Полуоткрытые диапазоны общеприняты в Python. Считается, что их использование более удобно и вызывает меньше программистских ошибок. Например, range(len(s)) порождает список индексов для списка s (частный случай – строки). В Python-последовательностях первый элемент имеет индекс 0. Для красивого вывода таблицы умножения применена операция форматирования % (для целых чисел тот же символ используется для обозначения операции взятия остатка от деления).

3.3. Обработка исключительных ситуаций

Для обработки особых ситуаций применяется знакомый механизм исключений. Лучше всего пояснить синтаксис оператора try-исхерт следующим примером:

```

# -*- coding: cp1251 -*-
try:
    a=float(input("Введите делимое: "))
    b=float(input("Введите делитель: "))
    print "Частное: ", a/b
except ZeroDivisionError:
    print "Деление на 0"
except:
    print "Ошибка"

```

В этом примере вводятся 2 числа и делятся одно на другое. В результате этих нехитрых действий может возникнуть несколько исключительных ситуаций, некоторые из

них отмечены в частях `except` (здесь использовано стандартное встроенное исключение Python – деление на ноль). Вторая часть `except` в этом примере улавливает все другие исключения, которые не были пойманы выше. Например, если введено хотя бы одно нечисловое значение, функция `float()` возбudit исключение `ValueError`. Его-то и сможет отловить последняя часть `except`. Разумеется, выполнение части `try` в случае возникновения ошибки уже не продолжается после выполнения одной из частей `except`.

Кроме описанной формы оператора, есть еще форма `try-finally` для гарантированного выполнения некоторых действий при передаче управления изнутри оператора `try-finally` вовне. Он может применяться для освобождения занятых ресурсов, или, например, для сохранения работы в файл перед крахом, что требует обязательного выполнения, независимо от произошедших внутри катаклизмов:

```
# -*- coding: cp1251 -*-
try:
    a=float("Полтора!")
finally:
    print "Обработка гарантированно завершена"
```

Все, что в ветке `finally`, выполнится обязательно. Разумеется, если и там не будет ошибки. Но вот смешивать вместе формы `try-except` и `try-finally`, к сожалению, нельзя.

3. ОБОРУДОВАНИЕ

ПЭВМ IBM PC, SVGA монитор с разрешением не менее 800*600 пикселей; клавиатура; мышь, ОС Linux/Windows, язык программирования Python, среда Geany

4. ПОДГОТОВКА К РАБОТЕ

1. Изучить теоретическую часть.

5. ПРОГРАММА РАБОТЫ

1. Ввести примеры программ в IDE Geany (см. теоретическую часть и приложение).
2. Отладить примеры программы, выполнить изменения в программах в отношении типов переменных, значений переменных и т.п.
3. Результаты работы программ показать преподавателю.
4. Оформить отчет.
4. Защитить лабораторную работу перед преподавателем.

6. СОДЕРЖАНИЕ ОТЧЕТА

1. Номер и название лабораторной работы
2. Цель и задачи
3. Текст программы
4. Результаты и выводы по лабораторной работе

Приложение к лабораторной работе № 1. ПРИМЕРЫ ПРОГРАММ

1. Типы данных

```
#!/usr/bin/env python
# coding: utf8

# Basic data types
a = 5                # int
b = 7.0              # float
c = 2>4              # boolean
d = "World"          # string
e = 1.5 + 0.5j        # complex

print type(a)
print type(b)
print type(c)
print type(d)
print type(e)

print a, b, c, d, e.real, e.imag
```

2. Основные операции

```
#!/usr/bin/env python
# coding: utf8

# *****
# Basic operations
# *****

a = 5                # int
b = 7.0              # float
c = 1 + 2            # 3
d = 5 - 3            # 2
e = a * b            # 35.0
f = 3.0 / 2          # 1.5
g = 3 / 2            # 1
h = 5 % 3            # 2
j = 10**7.3          # 19952623.1497

print type(a)
print type(b)
print type(c)
print type(d)
print type(e)
print type(f)
print type(g)
print type(h)
print type(j)

print a, b, c, d, e, f, g, h, j
```

3. Списки

```
#!/usr/bin/env python
# coding: utf8

# *****
# Lists
# *****

clouds = ["St", "Sc", "Ns", "As", "Ac"]

print type(clouds)
print clouds

clouds.append("Cu")
print clouds

clouds.append("Cb")
print clouds

clouds.sort()
print clouds

print clouds[2]
print clouds[2:5]
```

```
print clouds[2:]
print clouds[:5]
print clouds[2:5:2]
print clouds[2::2]
print clouds[:,2]
```

4. Массивы

```
#!/usr/bin/env python
# coding: utf8
```

```
*****
# Arrays
*****
```

```
A = zeros(12)
print A
```

```
A[0] = 1
A[1] = 2
A[11] = 12
print A
```

5. Цикл for

```
#!/usr/bin/env python
# coding: utf8
```

```
from numpy import *
```

```
*****
# Cycle "for"
*****
```

```
for i in range(10):
    print i
```

```
A = []
for i in range(10):
    A.append(i**i)
```

```
B = zeros(10, dtype=int)
for i in range(10):
    B[i] = i**i
```

```
print A
print B
```

6. Цикл while

```
#!/usr/bin/env python
# coding: utf8
```

```
from numpy import *
```

```
*****
# Cycle "while"
*****
```

```
i = 0
while i<10:
    print i
    i = i + 1
```

```
i = 0
A = []
while i<10:
    A.append(i**i)
    i = i + 1
```

```
print A
```

7. Печать данных по формату

```
#!/usr/bin/env python
# coding: utf8
```

```

*****
# Format print
*****

for i in range(10):
    A = i*18
    print "%02i\t%.1f" % (i, A)

```

8. Запись данных в файл

```

#!/usr/bin/env python
# coding: utf8

*****
# Write text file
*****

filename = "mydata.txt"

# Open the file
fd = open(filename, "w")

# Write values in file
for i in range(10):
    A = i*18
    fd.write("%i\t%.1f\n" % (i, A))

# Close file
fd.close()

```

9. Чтение из файла

```

#!/usr/bin/env python
# coding: utf8

from numpy import *
import csv
import sys

filename = "mydata.txt"

*****
# Read text file (version 1)
*****

# Load text file
data = loadtxt(filename)

# Print data
for i in range(len(data[:,0])):
    print data[i,:]

*****
# Read text file (version 2)
*****

# Open data file
fd = open(filename, "r")

# Read data
reader = csv.reader(fd, delimiter="\t")

# Print data
for row in reader:
    print row

```

10. Чтение данных из командной строки

```

#!/usr/bin/env python
# coding: utf8

from numpy import *
import csv
import sys
import os

```

```

# Read file name from command line
if len(sys.argv) < 2:
    sys.exit("Usage:\npython\t%s myfile.txt" % sys.argv[0])
if not os.path.exists(sys.argv[1]):
    sys.exit("ERROR: File %s was not found!" % sys.argv[1])
filename = sys.argv[1]

# Load text file
data = loadtxt(filename)

# Print data
for i in range(len(data[:,0])):
    print data[i,:]

```

11. Оператор условия if

```

#!/usr/bin/env python
# coding: utf8

```

```

*****
# Operator "if"
*****

for i in range(-50,51,10):
    if i<0:
        print "%i - C...PsP»PsPrPSPs" % (i)
    elif i>0:
        print "%i - C,PpPiP»Ps" % (i)
    else:
        print "%i - PSPsCЪPjP°P»CЪPSPs" % (i)

```

ЛАБОРАТОРНАЯ РАБОТА №2.

Структурное программирование на языке программирования Python. Часть 2.

1. ЦЕЛЬ И ЗАДАЧИ РАБОТЫ: Изучение основ структурного программирования на языке программирования Python.

2. ОБЩИЕ ПОЛОЖЕНИЯ

Начинать изучать язык программирования, неважно какой, необходимо с самых его основ, элементарных базовых операций и конструкций., изучить его работу с памятью. Анализировать и сравнивать реализации идентичных операций с уже известным вам языком. В данном случае Python часто будет сравниваться с Pascal.

По синтаксису они очень схожи, по крайней мере, гораздо больше чем с C++. Семантика этих языков чуть ли не одинаковая, а синтаксис большинства операторов различается незначительно. Python, кстати, даже немного более гибок в некоторых случаях, нежели Pascal. Например, функции могут возвращать не одно значение как в Pascal, а несколько. Наиважнейшее отличие этих языков кроется в их архитектуре, в способах хранения и обработки данных.

3. ОБОРУДОВАНИЕ

ПЭВМ IBM PC, SVGA монитор с разрешением не менее 800*600 пикселей; клавиатура; мышь, ОС Linux/Windows, язык программирования Python, среда Geany

4. ПОДГОТОВКА К РАБОТЕ

1. Изучить теоретическую часть (лабораторная работа №1).

5. ПРОГРАММА РАБОТЫ

1. Получить индивидуальные задания (см. Приложение А (к лабораторной работе №2))

Номер индивидуального задания соответствует последней цифре номера зачетки.

№ индивидуального задания	№№ задания 1 (см. приложение А)	№№ задания 2 (см. приложение А)	№№ задания 3 (см. приложение А)
0	1,2	15,1	1,2,3
1	2,3	14,3	4,5,6
2	4,5	10,11	7,8,9
3	6,7	12,13	10,11,12
4	8,9	14,15	13,14,15
5	10,11	1,2	16,17,18
6	12,13	2,3	19,20,5
7	14,15	4,5	1,10,20
8	15,1	6,7	3, 5,15,
9	14,3	8,9	11, 17, 19

2. Написать и отладить программы на языке Python.

3. Результаты работы программ показать преподавателю.
4. Оформить отчет.
5. Защитить лабораторную работу перед преподавателем.

6. СОДЕРЖАНИЕ ОТЧЕТА

1. Номер и название лабораторной работы
2. Цель и задачи
3. Текст программы
4. Результаты и выводы по лабораторной работе

Приложение А (к лабораторной работе №2)

ЗАДАНИЕ 1. Индивидуальные варианты.

$$1. t = \frac{2 \cos\left(x - \frac{\pi}{6}\right)}{0.5 + \sin^2 y} \left(1 + \frac{z^2}{3 - z^2/5}\right).$$

При $x=14.26$, $y=-1.22$, $z=3.5 \times 10^{-2}$ $t=0.564849$.

$$2. u = \frac{\sqrt[3]{8 + |x - y|^2 + 1}}{x^2 + y^2 + 2} - e^{|x-y|} (tg^2 z + 1)^x.$$

При $x=-4.5$, $y=0.75 \times 10^{-4}$, $z=0.845 \times 10^2$ $u=-55.6848$.

$$3. v = \frac{1 + \sin^2(x + y)}{\left|x - \frac{2y}{1 + x^2 y^2}\right|} x^{|y|} + \cos^2\left(\arctg \frac{1}{z}\right).$$

При $x=3.74 \times 10^{-2}$, $y=-0.825$, $z=0.16 \times 10^2$, $v=1.0553$.

$$4. w = |\cos x - \cos y|^{(1+2 \sin^2 y)} \left(1 + z + \frac{z^2}{2} + \frac{z^3}{3} + \frac{z^4}{4}\right).$$

При $x=0.4 \times 10^4$, $y=-0.875$, $z=-0.475 \times 10^{-3}$ $w=1.9873$.

$$5. \alpha = \ln\left(y^{-\sqrt{|x|}}\right) \left(x - \frac{y}{2}\right) + \sin^2 \arctg(z).$$

При $x=-15.246$, $y=4.642 \times 10^{-2}$, $z=20.001 \times 10^2$ $\alpha=-182.036$.

$$6. \beta = \sqrt{10(\sqrt[3]{x} + x^{y+2})} (\arcsin^2 z - |x - y|).$$

При $x=16.55 \times 10^{-3}$, $y=-2.75$, $z=0.15$ $\beta=-38.902$.

$$7. \gamma = 5 \arctg(x) - \frac{1}{4} \arccos(x) \frac{x + 3|x-y| + x^2}{|x-y|z + x^2}.$$

При $x=0.1722$, $y=6.33$, $z=3.25 \times 10^{-4}$ $\gamma=-172.025$.

$$8. \varphi = \frac{e^{|x-y|} |x-y|^{x+y}}{\arctg(x) + \arctg(z)} + \sqrt[3]{x^6 + \ln^2 y}.$$

При $x=-2.235 \times 10^{-2}$, $y=2.23$, $z=15.221$ $\varphi=39.374$.

$$9. \psi = \left| x^{\frac{y}{x}} - \sqrt[3]{\frac{y}{x}} \right| + (y-x) \frac{\cos y - \frac{z}{(y-x)}}{1 + (y-x)^2}.$$

При $x=1.825 \times 10^2$, $y=18.225$, $z=-3.298 \times 10^{-2}$ $\psi=1.2131$.

$$10. a = 2^{-x} \sqrt{x + \sqrt[4]{y}} \sqrt[3]{e^{x-1/\sin z}}.$$

При $x=3.981 \times 10^{-2}$, $y=-1.625 \times 10^3$, $z=0.512$ $a=1.26185$.

$$11. b = y^{\sqrt[3]{|x|}} + \cos^3(y) \frac{|x-y| \left(1 + \frac{\sin^2 z}{\sqrt{x+y}} \right)}{e^{|x-y|} + \frac{x}{2}}.$$

При $x=6.251$, $y=0.827$, $z=25.001$ $b=0.7121$.

$$12. c = 2^{(y^x)} + (3^x)^y - \frac{y \left(\arctg z - \frac{\pi}{6} \right)}{|x| + \frac{1}{y^2 + 1}}.$$

При $x=3.251$, $y=0.325$, $z=0.466 \times 10^{-4}$ $c=4.025$.

$$13. f = \frac{\sqrt[4]{y + \sqrt[3]{x-1}}}{|x-y|(\sin^2 z + \tg z)}.$$

При $x=17.421$, $y=10.365 \times 10^{-3}$, $z=0.828 \times 10^5$ $f=0.33056$.

$$14. g = \frac{y^{x+1}}{\sqrt[3]{|y-2|} + 3} + \frac{x + \frac{y}{2}}{2|x+y|} (x+1)^{-1/\sin z}.$$

При $x=12.3 \times 10^{-1}$, $y=15.4$, $z=0.252 \times 10^3$ $g=82.8257$.

$$15. h = \frac{x^{y+1} + e^{y-1}}{1 + x|y - \tg z|} (1 + |y-x|) + \frac{|y-x|^2}{2} - \frac{|y-x|^3}{3}.$$

При $x=2.444$, $y=0.869 \times 10^{-2}$, $z=-0.13 \times 10^3$ $h=-0.49871$.

ЗАДАНИЕ 2. Индивидуальные варианты.

1. Дана строка, состоящая из групп нулей и единиц. Посчитать количество нулей и единиц.
2. Посчитать в строке количество слов.
3. Найти количество знаков препинания в исходной строке.
4. Дана строка символов. Вывести на экран цифры, содержащиеся в строке.
5. Дана строка символов, состоящая из произвольных десятичных цифр, разделенных пробелами. Вывести вывести количество четных чисел в этой строке.
6. Дана строка символов, состоящая из произвольного текста на английском языке, слова разделены пробелами. Сформировать новую строку, состоящую из чисел длин слов в исходной строке.
7. Дана строка символов. Вывести на экран количество русских букв, входящих в эту строку.
8. Дана строка символов. Вывести на экран только строчные русские буквы, входящие в эту строку.
9. Дана строка символов, состоящая из произвольного текста на английском языке, слова разделены пробелами. В каждом слове заменить первую букву на цифру 1 и вывести преобразованную строку на экран.
10. Дана строка символов, состоящая из произвольного текста на английском языке, слова разделены пробелами. Удалить первую букву в каждом слове 1 и вывести преобразованную строку на экран.
11. Дана строка символов, состоящая из произвольного текста на английском языке, слова разделены пробелами. Поменять местами i- и j-ю буквы.
12. Дана строка символов, состоящая из произвольного текста на английском языке, слова разделены пробелами. Поменять местами первую и последнюю буквы каждого слова.
13. Дана строка символов, состоящая из произвольного текста на английском языке, слова разделены пробелами. Заменить все буквы латинского алфавита на знак '+’.
14. Дана строка символов, содержащая некоторый текст на русском языке. Заменить все большие буквы буквы 'А' на символ '*’.
15. Дана строка символов, состоящая из произвольного текста на английском языке, слова разделены пробелами. Поменять местами 1- и 3-ю буквы.

ЗАДАНИЕ 3. Индивидуальные варианты.

- 1) В массиве из 20 целых чисел найти наибольший элемент и поменять его местами с первым элементом.
- 2) В массиве из 10 целых чисел найти наименьший элемент и поменять его местами с последним элементом.
- 3) В массиве из 15 вещественных чисел найти наибольший элемент и поменять его местами с последним элементом.
- 4) В массиве из 25 вещественных чисел найти наименьший элемент и поменять его местами с первым элементом.
- 5) Дан массив X, содержащий 27 элементов. Вычислить и вывести элементы нового массива Y, где $y_i = 6.85x_i^2 - 1.52$. Если $y_i < 0$, то вычислить и вывести $a = x_i^3 - 0.62$ и продолжить вычисления; если $y_i \geq 0$, то вычислить и вывести $b = 1/x_i^2$ и продолжить вычисления.
- 6) Дан массив X, содержащий 16 элементов. Вычислить и вывести значения d_i , где $d_i = \frac{e^{x_i} + 2e^{-x_i}}{\sqrt{5 + \sin x_i}}$ и значения $d_i > 0.1$.
- 7) Дан массив Y, содержащий 25 элементов. Записать в массив R и вывести значения элементов, вычисляемые по формуле $r_i = \frac{5y_i + \cos^2 y_i}{2.35}$, $i=1,2,\dots,25$.

- 8) Дан массив F, содержащий 18 элементов. Вычислить и вывести элементы нового массива $p_i = 0.13f_i^3 - 2.5f_i + 8$. Вывести отрицательные элементы массива P.
- 9) Вычислить и вывести элементы массива Z, где $z_i = i^2 + 1$, если i – нечетное, и $z_i = 2i - 1$, если i – четное. Сформировать и вывести массив D: $d_i = 2.5z_i$, если $z_i < 2.5$ и $d_i = z_i / 2.5$, если $z_i \geq 2.5$.
- 10) Заданы массивы D и E. Вычислить и вывести значения $f_i = (2d_i + \sin e_i) / d_i$, где $i = 1, 2, \dots, 16$; вывести $1 < f_i < 3$.
- 11) В массиве R, содержащем 25 элементов, заменить значения отрицательных элементов квадратами значений, значения положительных увеличить на 7, а нулевые значения оставить без изменения. Вывести массив R.
- 12) Дан массив A целых чисел, содержащий 30 элементов. Вычислить и вывести сумму тех элементов, которые кратны 5.
- 13) Дан массив A целых чисел, содержащий 30 элементов. Вычислить и вывести сумму тех элементов, которые нечетны и отрицательны.
- 14) Дан массив A целых чисел, содержащий 30 элементов. Вычислить и вывести сумму тех элементов, которые удовлетворяют условию $|a_i| < i^2$.
- 15) Дан массив A целых чисел, содержащий 30 элементов. Вычислить и вывести количество и сумму тех элементов, которые делятся на 5 и не делятся на 7.
- 16) Дан массив A вещественных чисел, содержащий 25 элементов. Вычислить и вывести число отрицательных элементов и число членов, принадлежащих отрезку $[1, 2]$.
- 17) Дан массив C, содержащий 23 элемента. Вычислить и вывести среднее арифметическое всех значений $c_i > 3.5$.
- 18) Дан массив Z целых чисел, содержащий 35 элементов. Вычислить и вывести $R = S + P$, где S – сумма четных элементов, меньших 3, P – произведение нечетных элементов, больших 1.
- 19) Дан массив Q натуральных чисел, содержащий 20 элементов. Найти и вывести те элементы, которые при делении на 7 дают остаток 1, 2 или 5.
- 20) Дан массив Q натуральных чисел, содержащий 20 элементов. Найти и вывести те элементы, которые обладают тем свойством, что корни уравнения $q_i^2 + 3q_i - 5 = 0$ действительны и положительны.

ЛАБОРАТОРНАЯ РАБОТА №3.

Объекто-ориентированное программирование на языке
программирования Python. Часть 1.

ЦЕЛЬ И ЗАДАЧИ РАБОТЫ: Изучение основ объектно-ориентированного программирования на языке программирования Python.

ОБЩИЕ ПОЛОЖЕНИЯ

С точки зрения ООП, класс представляет собой коллекцию данных. Использование классов дает нам прежде всего преимущества абстрактного подхода в программировании.

1. **Полиморфизм:** в разных объектах одна и та же операция может выполнять различные функции. Слово «полиморфизм» имеет греческую природу и означает «имеющий многие формы». Простым примером полиморфизма может служить функция `count()`, выполняющая одинаковое действие для различных типов объектов: `'abc'.count('a')` и `[1, 2, 'a'].count('a')`. Оператор плюс полиморфичен при сложении чисел и при сложении строк.

2. **Инкапсуляция:** можно скрыть ненужные внутренние подробности работы объекта от окружающего мира. Это второй основной принцип абстракции. Он основан на использовании атрибутов внутри класса. Атрибуты могут иметь различные состояния в промежутках между вызовами методов класса, вследствие чего сам объект данного класса также получает различные состояния — `state`.

3. **Наследование:** можно создавать специализированные классы на основе базовых. Это позволяет нам избегать написания повторного кода.

4. **Композиция:** объект может быть составным и включать в себя другие объекты.

Объектно-ориентированный подход в программировании подразумевает следующий алгоритм действий.

1. Описывается проблема с помощью обычного языка с использованием понятий, действий, прилагательных.

2. На основе понятий формулируются классы.

3. На основе действий проектируются методы.

4. Реализуются методы и атрибуты.

Мы получили скелет — объектную модель. На основе этой модели реализуется наследование. Для проверки модели:

1) пишется так называемый `usecases` — сценарий возможного поведения модели, где проверяется вся функциональность;

2) функционал при этом может быть исправлен либо добавлен.

Объектно-ориентированный подход хорош там, где проект подразумевает долгосрочное развитие, состоит из большого количества библиотек и внутренних связей.

Механизм классов языка Python представляет собой смесь механизмов классов C++ и Modula-3. Наиболее важные особенности классов в питоне:

1) множественное наследование;

2) производный класс может переопределить любые методы базовых классов;

3) в любом месте можно вызвать метод с тем же именем базового класса;

4) все атрибуты класса в питоне по умолчанию являются `public`, т.е. доступны отовсюду; все методы — виртуальные, т.е. перегружают базовые.

Рассмотрим следующие аспекты объектно-ориентированного программирования.

1. Что такое класс.

2. Атрибуты класса.

3. `self`.

4. Наследование.

5. ООП в действии: пример создания классов.

1. Что такое класс

Класс — это пользовательский тип. Простейшая модель определения класса выглядит следующим образом:

```
class имя_класса:  
    инструкция 1  
    .  
    инструкция №
```

Каждая такая запись генерирует свой объект класса. Отличие от C++ в том, что в плюсах описание класса — это лишь объявление, а в питоне — это создание объекта. Есть также другой тип объекта — инстанс класса, который генерируется при вызове:

```
инстанс_класса = имя_класса()
```

Объект класса и инстанс класса — это два разных объекта. Первый генерируется на этапе объявления класса, второй — при вызове имени класса. Объект класса может быть один, инстансов класса может быть сколько угодно.

Инструкция — это, как правило, определение функции. При определении класса создается новое пространство имен и создается объект-класс, который является оболочкой для всех инструкций.

Объекты классов поддерживают два вида операций:

- доступ к атрибутам;
- создание экземпляра класса.

2. Атрибуты класса

Атрибуты класса бывают двух видов:

- атрибуты данных;
- атрибуты-методы.

Атрибуты данных обычно записываются сверху. Память для атрибутов выделяется в момент их первого присваивания — либо снаружи, либо внутри метода. Методы начинаются со служебного слова `def`.

Доступ к атрибутам выполняется по схеме `obj.attrname`.

Пример.

```
class Simple:  
    u'Простойкласс'  
    var = 87  
    def f(x):  
        return 'Hello world'
```

Здесь `Simple.var` и `Simple.f` — пользовательские атрибуты. Есть также стандартные атрибуты:

```
>>> print Simple.__doc__
```

Целое число

```
>>> print Simple.var.__doc__  
int(x[, base]) -> integer  
...
```

Создание экземпляра класса похоже на то, как будто мы делаем вызов функций:

```
smpl = Simple()
```

Будет создан пустой объект `smpl`. Если мы хотим, чтобы при создании выполнялись какие-то действия, нужно определить конструктор, который будет вызываться автоматически:

```
class Simple:
    def __init__(self):
        self.list = []
```

При создании объекта `smpl` будет создан пустой список `list`. Конструктору можно передать аргументы:

```
class Simple:
    def __init__(self, count, str):
        self.list = []
        self.count = count
        self.str = str

>>> s = Simple(1, '22')
>>> s.count, s.str
1 22
```

Атрибут данных можно сделать приватным (`private`) — т.е. недоступным снаружи — для этого слева нужно поставить два символа подчеркивания:

```
class Simple:
    u'Простой класс с приватным атрибутом'
    __private_attr = 10
    def __init__(self, count, str):
        self.__private_attr = 20
        print self.__private_attr

s = Simple(1, '22')
print s.__private_attr
```

Последняя строка вызовет исключение — атрибут `__private_attr` годен только для внутреннего использования.

Методы необязательно определять внутри тела класса:

```
def method_for_simple(self, x, y):
    return x + y

class Simple:
    f = method_for_simple

>>> s = Simple()
>>> print s.f(1, 2)
3
```

Пустой класс можно использовать в качестве заготовки для структуры данных:

```
class Customer:
    pass
```

```
custom = Customer()
custom.name = 'Вася'
custom.salary = 100000
```

3. self

Обычно первый аргумент в имени метода — self. Как говорит автор языка Гвидо Ван Россум, это не более чем соглашение: имя self не имеет абсолютно никакого специального значения.

self полезен для того, чтобы обращаться к другим атрибутам класса:

```
class Simple:
    def __init__(self):
        self.list = []
    def f1(self):
        self.list.append(123)
    def f2(self):
        self.f1()
```

```
>>> s = Simple()
>>> s.f2()
>>> print s.list
[123]
```

Self — это аналог "this" в C++.

4. Наследование

Определение производного класса выглядит следующим образом:

```
class Derived(Base):
```

Если базовый класс определен не в текущем модуле:

```
class Derived(module_name.Base):
```

Разрешение имен атрибутов работает сверху вниз: если атрибут не найден в текущем классе, поиск продолжается в базовом классе, и так далее по рекурсии. Производные классы могут переопределить методы базовых классов — все методы являются в этом смысле виртуальными. Вызвать метод базового класса можно с префиксом:

```
Base.method()
```

В питоне существует ограниченная поддержка множественного наследования:

```
class Derived(Base1, Base2, Base3):
```

Поиск атрибута производится в следующем порядке:

1. в Derived;
2. в Base1, затем рекурсивно в базовых классах Base1;
3. в Base2, затем рекурсивно в базовых классах Base2
4. и т.д.

5. ОБОРУДОВАНИЕ

ПЭВМ IBMPC, VGA монитор с разрешением не менее 800*600 пикселей; клавиатура; мышь, ОС Linux/Windows, язык программирования Python, среда Geany

6. ПОДГОТОВКА К РАБОТЕ
2. Изучить теоретическую часть.

7. ПРОГРАММА РАБОТЫ

5. Отладить примеры, приведенные в общих положениях данной лабораторной работы.
6. Написать и отладить следующую программу на Python.
Создать два класса: `Person` — хранит общую информацию о людях — имя, профессия, зарплата; класс `Manager` — специализированный производный класс. В классе `Person` создать свою версию для стандартной встроенной функции `str`, которая есть по умолчанию в любом питоновском классе.

Часть программы представлена ниже:

```
# -*- coding: utf-8 -*-
class Person:
    def __init__(self, name, job=None, pay=0):
        self.name = name
        self.job = job
        self.pay = pay
    def lastName(self):
        return self.name.split()[-1]
    def giveRaise(self, percent):
        self.pay = int(self.pay * (1 + percent))
    def __str__(self):
        return 'Person: %s, %s' % (self.name, self.pay)

class Manager(Person):
    def __init__(self, name, pay):
        self.giveRaise(self, percent=100)
        Person.giveRaise(self, percent + bonus)

Создаем первый инстанс класса Person:
>>>ivan = Person('Иван Petrov')

Создаем второй инстанс класса Person:
>>>john = Person('John Sidorov', job='dev', pay=100000)

Вызываем перегруженную функцию __str__:
>>>print(ivan)
print(tom)
```

7. Результаты работы программ показать преподавателю.
8. Оформить отчет.
9. Защитить лабораторную работу перед преподавателем.

6. СОДЕРЖАНИЕ ОТЧЕТА

5. Номер и название лабораторной работы
6. Цель и задачи
7. Текст программы
8. Результаты и выводы по лабораторной работе

ЛАБОРАТОРНАЯ РАБОТА № 4

Объекто-ориентированное программирование на языке

программирования Python. Часть 2.

ЦЕЛЬ И ЗАДАЧИ РАБОТЫ: Изучение основ объектно-ориентированного программирования на языке программирования Python (специальные методы и атрибуты классов)

ОБЩИЕ ПОЛОЖЕНИЯ

1. Объекты классов и специальные методы

Объект-класс создается с помощью определения класса. Объекты-классы имеют следующие атрибуты:

- `__name__` — имя класса;
- `__module__` — имя модуля;
- `__dict__` — словарь атрибутов класса, можно изменять этот словарь напрямую;
- `__bases__` — кортеж базовых классов в порядке их следования;
- `__doc__` — строка документации класса.

2. Экземпляры классов и специальные методы

Экземпляр (инстанс) класса возвращается при вызове объекта-класса. Объект у класса может быть один, экземпляров (или инстансов) — несколько. Экземпляры имеют следующие атрибуты:

- `__dict__` — словарь атрибутов класса, можно изменять этот словарь напрямую;
- `__class__` — объект-класс, экземпляром которого является данный инстанс;
- `__init__` — конструктор. Если в базовом классе есть конструктор, конструктор производного класса должен вызвать его;
- `__del__` — деструктор. Если в базовом классе есть деструктор, деструктор производного класса должен вызвать его;
- `__cmp__` — вызывается для всех операций сравнения;
- `__hash__` — возвращает хеш-значение объекта, равное 32-битному числу;
- `__getattr__` — возвращает атрибут, недоступный обычным способом;
- `__setattr__` — присваивает значение атрибуту;
- `__delattr__` — удаляет атрибут;
- `__call__` — срабатывает при вызове экземпляра класса.

3. Экземпляры классов в качестве последовательностей

Экземпляры классов можно использовать для эмуляции последовательностей. Для такой реализации есть встроенные методы:

- `__len__` — возвращает длину последовательности;
- `__getitem__` — получение элемента по индексу или ключу;
- `__setitem__` — присваивание элемента с данным ключом или индексом;
- `__delitem__` — удаление элемента с данным ключом или индексом;
- `__getslice__` — возвращает вложенную последовательность;
- `__setslice__` — заменяет вложенную последовательность;
- `__delslice__` — удаляет вложенную последовательность;
- `__contains__` — реализует оператор `in`.

4. Приведение объектов к базовым типам

Объекты классов можно привести к строковому или числовому типу.

- `__repr__` — возвращает формальное строковое представление объекта;
- `__str__` — возвращает строковое представление объекта;

`__oct__` , `__hex__` , `__complex__` , `__int__` , `__long__` , `__float__` — возвращают строковое представление в соответствующей системе счисления.

5. Bound и unbound методы

Рассмотрим конкретный пример. Есть базовый класс `Cat`, и есть производный от него класс `Barsik`:

```
class Cat:
    def __init__(self):
        self.hungry = True
    def eat(self):
        if self.hungry:
            print 'I am hangry...'
            self.hungry = False
        else:
            print 'No, thanks!'

class Barsik(Cat):
    def __init__(self):
        self.sound = 'Aaaamm!'
        print self.sound
```

Создаем экземпляр производного класса:

```
>>> brs = Barsik()
Aaaamm!
>>> brs.eat()
AttributeError: Barsik instance has no attribute 'hungry'
```

На первый взгляд — странная ошибка, поскольку атрибут `hungry` есть в базовом классе. На самом деле, конструктор производного класса — перегруженный, при этом конструктор базового класса не вызывается, и его нужно явно вызвать. Это можно сделать двумя путями. Первый вариант считается устаревшим:

```
class Barsik(Cat):
    def __init__(self):
        Cat.__init__(self)
        self.sound = 'Aaaamm!'
        print self.sound
```

Здесь мы напрямую вызываем конструктор базового класса, не создавая инстанс базового класса `Cat` — поэтому такой базовый конструктор относится к категории unbound-методов, в пику методам, которые вызываются для инстансов классов и называются bound-методами. Для вызова bound-метода в качестве первого параметра методу нужно передать инстанс класса.

6. Метод super

Второй вариант: в начале программы нужно определить метакласс , который указывает на то, что класс реализован в так называемом новом стиле — new-style. Затем нужно вызвать стандартный метод `super` для базового конструктора:

```
__metaclass__ = type
...
class Barsik(Cat):
```

```
def __init__(self):
    super(Barsik, self).__init__()
    self.sound = 'Aaaammm!'
    printself.sound

>>>brs = Barsik()
>>>brs.eat()
Aaaammm!
I am hangry...
```

7. Статические методы

Статический метод — функция, определенная вне класса и не имеющая атрибута `self`:

```
class Spam:
    numInstances = 0
    def __init__(self):
        Spam.numInstances = Spam.numInstances + 1

    def printNumInstances():
        print "Number of instances created: ", Spam.numInstances

>>> a=Spam()
>>> b=Spam()
>>> printNumInstances()
Numberofinstancescreated: 2
```

Статический метод может быть определен и внутри класса — для этого используется ключевое слово `staticmethod`, причем метод может быть вызван как статически, так и через инстанс:

```
class Multi:
    def meth(self, x):
        print self, x
    def smeth(x):
        print x
    def cmeth(cls, x):
        print cls, x
    smeth = staticmethod(smeth)
    cmeth = classmethod(cmeth)

>>> Multi.smeth(3)
3
>>> obj=Multi()
>>> obj.smeth(5)
5
```

Методы класса определяются с помощью ключевого слова `classmethod` — здесь автоматически питон передает в качестве первого параметра сам класс (`cls`):

```
>>> Multi.cmeth(7)
__main__.Multi 7
>>> obj.cmeth(10)
```

8. Итератор

Итераторы хороши там, где списки не подходят в силу того, что занимают много памяти, а итератор возвращает его конкретное значение. В классе нужно определить два стандартных метода — `__iter__` и `next`. Метод `__iter__` будет возвращать объект через метод `next`:

```
class Reverse:
    def __init__(self, data):
        self.data = data
        self.index = len(data)
    def __iter__(self):
        return self
    def next(self):
        if self.index == 0:
            raise StopIteration
        self.index = self.index - 1
        return self.data[self.index]

>>> for char in Reverse('12345'):
>>> print char
5
4
3
2
1
```

Итератор можно сконвертировать в список:

```
>>> rvr = list(Reverse('12345'))
>>> rvr
['5', '4', '3', '2', '1']
```

9. Property

Property — атрибут класса, возвращаемый через стандартную функцию `property`, которая в качестве аргументов принимает другие функции класса:

```
class DateOffset:
    def __init__(self):
        self.start = 0

    def _get_offset(self):
        self.start += 5
        return self.start

    offset = property(_get_offset)

>>> d = DateOffset()
>>> d.offset
5
>>> d.offset
```

10. Singleton

Данный паттерн позволяет создать всего один инстанс для класса. Используется метод `__new__`:

```
class Singleton(object):
    def __new__(cls, *args, **kw):
        if not hasattr(cls, '_instance'):
            orig = super(Singleton, cls)
            cls._instance = orig.__new__(cls, *args, **kw)
        return cls._instance

>>> one = Singleton()
>>> two = Singleton()
>>> id(one)
3082687532
>>> id(two)
3082687532
```

11. Слоты

Слоты — это список атрибутов, задаваемый в заголовке класса с помощью `__slots__`. В инстансе необходимо назначить атрибут, прежде чем пользоваться им:

```
class limiter(object):
    __slots__ = ['age', 'name', 'job']

>>> x = limiter()
>>> x.age = 20
```

12. Функтор

Функтор — это класс, имеющий метод `__call__` — при этом объект можно вызывать как функцию.

Пример. Пусть у нас имеется класс `Person`, имеется коллекция объектов этого класса — `people`, нужно отсортировать эту коллекцию по фамилиям. Для этого можно использовать функтор `Sortkey`:

```
class SortKey:
    def __init__(self, *attribute_names):
        self.attribute_names = attribute_names

    def __call__(self, instance):
        values = []
        for attribute_name in self.attribute_names:
            values.append(getattr(instance, attribute_name))
        return values

class Person:
    def __init__(self, forename, surname, email):
        self.forename = forename
        self.surname = surname
        self.email = email
```

```

>>>people=[]
>>> p=Person('Petrov',"")
>>>people.append(p)
>>> p=Person('Sidorov',"")
>>>people.append(p)
>>> p=Person(u'Ivanov',"")
>>>people.append(p)
>>>for p in people:
... printp.forename
Petrov
Sidorov
Ivanov
>>>people.sort(key=SortKey("forename"))
>>>for p in people:
... printp.forename
Ivanov
Petrov
Sidorov

```

13. Дескриптор

Дескриптор — это класс, который хранит и контролирует атрибуты других классов. Вообще любой класс, который имплементирует один из специальных методов — `__get__`, `__set__`, `__delete__`, является дескриптором.

Пример:

```

class ExternalStorage:
    __slots__ = ("attribute_name",)
    __storage = {}

    def __init__(self, attribute_name):
        self.attribute_name = attribute_name

    def __set__(self, instance, value):
        self.__storage[id(instance), self.attribute_name] = value

    def __get__(self, instance, owner=None):
        if instance is None:
            return self
        return self.__storage[id(instance), self.attribute_name]

class Point:
    __slots__ = ()
    x = ExternalStorage("x")
    y = ExternalStorage("y")
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

>>> p1=Point(1,2)
>>> p2=Point(3,4)

```

В данном случае класс Point не имеет собственных атрибутов x, y, хотя вызывает их

так, как будто они есть — на самом деле они хранятся в дескрипторе ExternalStorage.

14. Sequence

Последовательность реализуется с помощью методов `__getitem__`, `__setitem__`. В данном примере класс `MySequence` возвращает по индексу элемент последовательности неопределенной длины, представляющей собой арифметическую прогрессию вида: 1 3 5 7 ...Здесь нельзя применить стандартные методы `__del__`, `__len__`:

```
class MySequence:
    def __init__(self, start=0, step=1):
        self.start = start
        self.step = step
        self.changed = {}
    def __getitem__(self, key):
        return self.start + key*self.step
    def __setitem__(self, key, value):
        self.changed[key] = value

>>> s = MySequence(1,2)
>>> s[0]
1
>>> s[1]
3
>>> s[100]
201
```

7. ОБОРУДОВАНИЕ

ПЭВМ IBMPC, VGA монитор с разрешением не менее 800*600 пикселей; клавиатура; мышь, ОС Linux/Windows, язык программирования Python, среда Geany

8. ПОДГОТОВКА К РАБОТЕ

Изучить теоретическую часть.

7. ПРОГРАММА РАБОТЫ

1. Написать и отладить программу на Python, используя понятия и методы объектно-ориентированного программирования (индивидуальные задания из лабораторной работы №2).
2. Результаты работы программ показать преподавателю.
3. Оформить отчет.
4. Защитить лабораторную работу перед преподавателем.

6. СОДЕРЖАНИЕ ОТЧЕТА

1. Номер и название лабораторной работы
2. Цель и задачи
3. Текст программы
4. Результаты и выводы по лабораторной работе

ЛАБОРАТОРНАЯ РАБОТА №5.

Функциональное программирование на языке программирования Python. Часть 1

ЦЕЛЬ И ЗАДАЧИ РАБОТЫ: Изучение основ функционального программирования на языке программирования Python.

1. ОБЩИЕ ПОЛОЖЕНИЯ

Функции являются **абстракциями**, в которых детали реализации некоторого действия скрываются за отдельным именем. Хорошо написанный набор функций позволяет использовать их много раз. Стандартная библиотека Python содержит множество готовых и отлаженных функций, многие из которых достаточно универсальны, чтобы работать с широким спектром входных данных. Даже если некоторый участок кода не используется несколько раз, но по входным и выходным данным он достаточно автономен, его смело можно выделить в отдельную функцию.

Функциональное программирование - это стиль программирования, использующий только композиции функций. Другими словами, это программирование в выражениях, а не в императивных командах.

Как отмечает Дэвид Мертц (David Mertz) в своей статье о функциональном программировании на Python, "функциональное программирование - программирование на функциональных языках (LISP, ML, OCAML, Haskell, ...)", основными атрибутами которых являются:

- "Наличие функций первого класса (функции наравне с другими объектами можно передавать внутрь функций).
- Рекурсия является основной управляющей структурой в программе. Обработка списков (последовательностей).
- Запрещение побочных эффектов у функций, что в первую очередь означает отсутствие присваивания (в "чистых" функциональных языках)
- Запрещение операторов, основной упор делается на выражения. Вместо операторов вся программа в идеале - одно выражение с сопутствующими определениями.

Ключевой вопрос: **что** нужно вычислить, а не **как**.

Использование функций более высоких порядков (функции над функциями над функциями)".

1) Функциональная программа

В математике функция отображает объекты из одного множества (множества определений функции) в другое (множество значений функции).

Математические функции (их называют **чистыми**) "механически", однозначно вычисляют результат по заданным аргументам. Чистые функции не должны хранить в себе какие-либо данные между двумя вызовами. Их можно представлять себе черными ящиками, о которых известно только то, что они делают, но совсем не важно, как.

Программы в функциональном стиле конструируются как **композиция** функций. При этом функции понимаются почти так же, как и в математике: они отображают одни объекты в другие. В программировании "чистые" функции - идеал, не всегда достижимый на практике. Практически полезные функции обычно имеют **побочный эффект**: сохраняют состояние между вызовами или меняют состояние других объектов. Например, без побочных эффектов невозможно представить себе функции ввода-вывода. Собственно, такие функции ради этих "эффектов" и используются. Кроме того, математические функции легко работают с объектами, требующими бесконечного объема информации (например, вещественные числа). В общем случае компьютерная программа может выполнить лишь приближенные вычисления.

Кстати, бинарные операции "+", "-", "*", "/", которые записываются в выражениях, являются "математическими" функциями над двумя аргументами -- **операндами**. Их используют настолько часто, что синтаксис языка программирования имеет для них более короткую запись. Модуль `operator` позволяет представлять эти операции в функциональном стиле:

```
>>> from operator import add, mul
>>> print add(2, mul(3, 4))
```

14

2) Функция: определение и вызов

Как уже говорилось, определить функцию в Python можно двумя способами: с помощью оператора `def` и `lambda`-выражения. Первый способ позволяет использовать операторы. При втором - определение функции может быть только выражением.

Методы классов определяются так же, как и функции. Отличие состоит в специальном смысле первого аргумента `self` (в нем передается экземпляр класса).

Лучше всего рассмотреть синтаксис определения функции на нескольких примерах. После определения соответствующей функции показан один или несколько вариантов ее вызова (некоторые примеры взяты из стандартной библиотеки).

Определение функции должно содержать список **формальных параметров** и **тело определения функции**. В случае с оператором `def` функции также задается некоторое имя. Формальные параметры являются локальными именами внутри тела определения функции, а при вызове функции они оказываются связанными с объектами, переданными как фактические параметры. Значения по умолчанию вычисляются в момент выполнения оператора `def`, и потому в них можно использовать видимые на момент определения имена.

Вызов функции синтаксически выглядит как объект-функция(фактически параметры). Обычно объект-функция - это просто имя функции, хотя это может быть и любое выражение, которое в результате вычисления дает исполняемый объект.

Функция одного аргумента:

```
def swapcase(s):
    return s.swapcase()
print swapcase("ABC")
```

Функция двух аргументов, один из которых необязателен и имеет значение по умолчанию:

```
def inc(n, delta=1):
    return n+delta
print inc(12) print
inc(12, 2)
```

Функция с одним обязательным аргументом, с одним, имеющим значение по умолчанию и неопределенным числом именованных аргументов:

```
def wrap(text, width=70, **kwargs): from
textwrap import TextWrapper
```

```
# kwargs - словарь с именами и значениями аргументов w =
TextWrapper(width=width, **kwargs)
return w.wrap(text)
print wrap("my long text ...", width=4)
```

Функция произвольного числа аргументов:

```
def max_min(*args):
    # args - список аргументов в порядке их указания при вызове return
max(args), min(args)
print max_min(1, 2, -1, 5, 3)
```

Функция с обычными (позиционными) и именованными аргументами:

```
def swiss_knife(arg1, *args, **kwargs): print
arg1
print args print
kwargs return None
print swiss_knife(1)
print swiss_knife(1, 2, 3, 4, 5)
print swiss_knife(1, 2, 3, a='abc', b='sdf')

# print swiss_knife(1, a='abc', 3, 4)          # !!! ошибка

lst = [2, 3, 4, 5]

dct = {'a': 'abc', 'b': 'sdf'} print
swiss_knife(1, *lst, **dct)
```

Пример определения функции с помощью lambda -выражения дан ниже:

```
func = lambda x, y: x + y
```

В результате lambda -выражения получается безымянный объект-функция, которая

затем используется, например, для того, чтобы связать с ней некоторое имя. Однако, как правило, определяемые `lambda` -выражением функции, применяются в качестве параметров функций.

В языке Python функция может вернуть только одно значение, которое может быть кортежем. В следующем примере видно, как стандартная функция `divmod()` возвращает частное и остаток от деления двух чисел:

```
def bin(n):
    """Цифры двоичного представления натурального числа """
    digits = []
    while n > 0:
        n, d = divmod(n, 2)
        digits = [d] + digits
    return digits
print bin(69)
```

Примечание:

Важно понять, что за именем функции стоит объект. Этот объект можно связать с другим именем:

```
def add(x, y):
    return x + y

addition = add # теперь addition и add - разные имена одного и того же объекта
```

Пример, в котором в качестве значения по умолчанию аргумента функции используется изменяемый объект (список). Этот объект - один и тот же для всех вызовов функций, что может привести к казусам:

```
def mylist(val, lst=[]):
    lst.append(val)
    return lst

print mylist(1), print mylist(2)
```

Вместо ожидаемого `[1] [2]` получается `[1] [1, 2]`, так как добавляются элементы к "значению по умолчанию".

Правильный вариант решения будет, например, таким:

```
def mylist(val, lst=None):
    lst = lst or []
    lst.append(val)
    return lst
```

Конечно, приведенная выше форма может использоваться для хранения в функции некоторого состояния между ее вызовами, однако, практически всегда вместо функции с таким побочным эффектом лучше написать класс и использовать его экземпляр.

3) Рекурсия

В некоторых случаях описание функции элегантнее всего выглядит с применением вызова этой же функции. Такой прием, когда функция вызывает саму себя, называется **рекурсией**. В функциональных языках рекурсия обычно используется много чаще, чем итерация (циклы).

В следующем примере переписывается функция `bin()` в рекурсивном варианте:

```
def bin(n):
    """Цифры двоичного представления натурального числа """
    if n == 0: return []
    n, d = divmod(n, 2) return
bin(n) + [d]
print bin(69)
```

Здесь видно, что цикл `while` больше не используется, а вместо него появилось условие окончания рекурсии: условие, при выполнении которого функция не вызывает себя.

Конечно, в погоне за красивым рекурсивным решением не следует упускать из виду эффективность реализации. В частности, пример реализации функции для вычисления n -го числа Фибоначчи это демонстрирует:

```
def Fib(n): if n
< 2:
return n else:
return Fib(n-1) + Fib(n-2)
```

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

Во многих языках программирования `map` реализуют через рекурсию. Это касается и Python

```
r_map = lambda fn, arg: pattern_match((
    (hd_tl_partial(identity, is_blank), lambda arr: [fn(arr[0])]), # x::[] -> fn(x)
    (
        hd_tl_partial(identity, identity),
        lambda arr: [fn(arr[0])] r_map(fn, arr[1]) # x::xs -> fn(x)::r_map(fn, xs)
    ),
), arg)

print(r_map(lambda x: x**2, range(10))) # [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Сейчас каррируем:

```
r_map_curry = curry_n(r_map, 2)
```

```
twice = r_map_curry(lambda x: x * 2)
print(twice(range(10)))
try:
print(twice(range(1000))) # [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
except RuntimeError as e:
print(e) # maximum recursion depth exceeded in comparison
```

Испробуем хвостовую рекурсию.

Для этого выполним «проверку» на None:

```
is_none = match_curry(lambda obj: obj is None)
```

И проверки пары:

```
pair = lambda match_x, match_y: lambda arr: (match_x(arr[0]), match_y(arr[1]))
```

И сам map:

```
def r_map_tail(fn, arg):
    aux = lambda arg: pattern_match((
        (pair(identity, is_none), lambda arr: aux([arr[0], []])), # если аккумулятор None,
        делаемое []
        (
            pair(hd_tl_partial(identity, is_blank), identity),
            lambda arr: arr[1] [fn(arr[0][0])] # если (x::[], acc), то прибавляем к аккумулятору fn(x) и
            возвращаем его
        ),
        (
            pair(hd_tl_partial(identity, identity), identity),
            lambda arr: aux([arr[0][1], arr[1] [fn(arr[0][0])]]) # если (x::xs, acc),
            тогда делаем рекурсивный вызов с xs и аккумулятором fn(x)
        ),
    ), arg)
    return aux([arg, None])
```

Рассмотрим еще один пример.

```
r_map_tail_curry = curry_n(r_map_tail, 2)
twice_tail = r_map_tail_curry(lambda x: x * 2)
print(twice_tail(range(10))) # [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
try:
print(twice_tail(range(10000)))
except RuntimeError as e:
print(e) # maximum recursion depth exceeded
```

Внимание! Python не оптимизирует хвостовую рекурсию! Но это можно обойти следующим способом:

```
def tail_fnc(fn):
    called = False
    calls = []

    def run():
        while len(calls): # вызываем функцию с доводами из списка
            res = fn(*calls.pop())
        return res
```

```

def call(*args):
    nonlocal called
    calls.append(args) # добавляем доводы в список
    if not called: # проверяем вызвалась ли функция, если нет - запускаем цикл
        called = True
    return run()
return call

```

Реализуем с этим map:

```

def r_map_really_tail(fn, arg):
    aux = tail_fnc(lambda arg: pattern_match(( # декорируем вспомогательную функцию
        (pair(identity, is_none), lambda arr: aux([arr[0], []])), # если аккумулятор None,
        делаемого []
        (
            pair(hd_tl_partial(identity, is_blank), identity),
            lambda arr: arr[1] [fn(arr[0][0])] # если (x::[], acc), то прибавляем к аккумулятору fn(x) и
            возвращаем его
        ),
        (
            pair(hd_tl_partial(identity, identity), identity),
            lambda arr: aux([arr[0][1], arr[1] [fn(arr[0][0])]]) # если (x::xs, acc),
            тогда делаем рекурсивный вызов с xs и аккумулятором fn(x)
        ),
    ), arg))
    return aux([arg, None])

r_map_really_tail_curry = curry_n(r_map_really_tail, 2)
twice_really_tail = r_map_really_tail_curry(lambda x: x * 2)
print(twice_really_tail(range(1000))) # [0, 2, 4, 6, 8, 10, 12, 14, 16, 18...]

```

Рассмотрим ещё один пример использования map.

```

def tail_r_map(fn, arr_):
    @tail_fnc
    def aux(arr, acc=None):
        x, *xs = arr
        if xs:
            return aux(xs, acc [fn(x)])
        else:
            return acc [fn(x)]
        return aux(arr_, [])

curry_tail_r_map = curry_2(tail_r_map)

```

И сделаем на нём умножение всех нечетных чисел в списке на 2:

```

@curry_tail_r_map
def twice_if_odd(x):
    if x % 2 == 0:
        return x * 2
    else:
        return x

```

```
print(twice_if_odd(range(10000))) # [0, 1, 4, 3, 8, 5, 12, 7, 16, 9, 20, 11, 24, 13, 28, 15, 32, 17, 36, 19...
```

Сравним эффективность различных вариантов map:

```
from time import time
```

```
checker = lambda x: x ** 2 - x
limit = 100000
```

```
start = time()
xs = [checker(x) for x in range(limit)][::-1]
print('inline for:', time() - start)
```

```
start = time()
xs = list(map(checker, xs))[::-1]
print('map:', time() - start)
```

```
calculate = curry_tail_r_map(checker)
start = time()
xs = calculate(xs)[::-1]
print('r_map without pattern matching:', time() - start)
```

```
calculate = r_map_really_tail_curry(checker)
start = time()
xs = calculate(xs)[::-1]
print('r_map with pattern matching:', time() - start)
```

В результате получим:

inline for: 0.010764837265014648

map: 0.010544061660766602

r_map without pattern matching: 4.720803737640381

r_map with pattern matching: 8.376755237579346

Предупреждение:

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

4) Функции как параметры и результат

Как уже не раз говорилось, 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, называют **функциями высшего порядка**. В Python функции высшего порядка применяются программистами достаточно часто. 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.

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
>>> max(*lst)
3

>>> apply(dict, [], dct) {'a':
4, 'b': 5}

>>> dict(**dct)
{'a': 4, 'b': 5}
```

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, кортежи, словари.

Функции `range()` и `xrange()`

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

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()`. Результатом будет 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 `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)]
```

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.')
'HithereIameatinganapple'
```

Списковые включения

Для более естественной записи обработки списков в Python 2 была внесена новинка: списковые включения. Фактически это специальный сокращенный синтаксис для вложенных циклов `for` и условий `if`, на самом низком уровне 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 следующей таблице приведены эквивалентные выражения в разных формах:

В форме функции	В форме спискового включения
<code>filter(f, lst)</code>	<code>[x for x in lst if f(x)]</code>
<code>filter(None, lst)</code>	<code>[x for x in lst if x]</code>
<code>map(f, lst)</code>	<code>[f(x) for x in lst]</code>

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()`.

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)
```

Совет:

Следует помнить, что в качестве передаваемого объекта может оказаться список, который позволит накапливать промежуточные результаты. Тем самым, reduce() может использоваться для генерации последовательностей.

В следующем примере накапливаются промежуточные результаты суммирования:

```
lst = range(10)  
f = lambda x, y: (x[0] + y, x[1]+[x[0] + y])  
reduce(f, lst, (0, []))
```

В итоге получается:

```
(45, [0, 1, 3, 6, 10, 15, 21, 28, 36, 45])
```

Функция zip()

Эта функция возвращает список кортежей, в котором i -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')]
```

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, промежуточные данные не будут требовать много места для хранения - ведь 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, внутренне оператор for запрашивает от последовательности ее итератор. Объект файлового типа тоже (построчный) итератор, что позволяет обрабатывать большие файлы, не считывая их целиком в память.

Там, где требуется итератор, можно использовать последовательность.

Работа с итераторами рассматривается в разделе, посвященном функциональному программированию, так как итераторами удобно манипулировать именно в функциональном стиле.

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, 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.

Функция `iter()`

Эта функция 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 породить 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]
```

Примечание:

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

Функция `enumerate()`

Эта функция создает итератор, нумерующий элементы другого итератора. Результирующий итератор выдает кортежи, в которых 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')]
```

Функция `sorted()`

Эта функция, появившаяся в Python 2.4, позволяет создавать итератор, выполняющий сортировку:

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

Далее рассматриваются функции модуля `itertools`.

Функция `itertools.chain()`

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

даст в результате
1 2 3 8 9 0

Аналогами `map()` и `filter()` в модуле `itertools` являются `imap()` и `ifilter()`.

Отличие `imap()` от `map()` в том, что вместо значения от преждевременно завершившихся итераторов объект `None` не подставляется. Пример:

```
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 итерируемыми (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,  
  
...  
1a 2b
```

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

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

Функции `itertools.takewhile()` и `itertools.dropwhile()`

Некоторую новизну вносит другой вид фильтра: `takewhile()` и его "отрицательный" аналог `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,  
1 -2 3 -3
```

Таким образом, `takewhile()` дает значения, пока условие истинно, а остальные

значения даже не берет из итератора (именно не берет, а не высасывает все до конца!). И, наоборот, `dropwhile()` ничего не выдает, пока выполняется условие, зато потом выдает все без остатка.

Функция `itertools.izip()`

Функция `izip()` аналогична встроенной `zip()`, но не тратит много памяти на построение списка кортежей, так как итератор выдает их строго по требованию.

Функция `itertools.groupby()`

Эта функция дебютировала в Python 2.4. Функция принимает два аргумента: итератор (обязательный) и необязательный аргумент - функцию, дающую значение ключа: `groupby(iterable[, func])`. Результатом является итератор, который возвращает двухэлементный кортеж: ключ и итератор по идущим подряд элементам с этим ключом. Если второй аргумент опущен, элемент итератора сам является ключом. В следующем примере группируются идущие подряд положительные и отрицательные элементы:

```
import itertools, math
lst = map(lambda x: math.sin(x*.4), range(30))

for k, i in itertools.groupby(lst, lambda x: x > 0): print k, lst(i)
```

Функция `itertools.tee()`

Эта функция тоже появилась в Python 2.4. Она позволяет клонировать итераторы. Первый аргумент - итератор, подлежащий клонированию. Второй (`N`) -- количество необходимых копий. Функция возвращает кортеж из `N` итераторов. По умолчанию `N=2`. Функция имеет смысл, только если итераторы задействованы более или менее параллельно. В противном случае выгоднее превратить исходный итератор в список.

Собственный итератор

Для полноты описания здесь представлен пример итератора, определенного пользователем. Если пример не очень понятен, можно вернуться к нему после изучения объектно-ориентированного программирования:

```
class Fibonacci:

    """Итератор последовательности Фибоначчи до N"""

    def __init__(self, N):

        self.n, self.a, self.b, self.max = 0, 0, 1, N

    def __iter__(self):

        # сами себе итератор: в классе есть метод next() return self

    def next(self):

        if self.n < self.max:
```

```

a, self.n, self.a, self.b = self.a, self.n+1, self.b, self.a+self.b

return a else:

raise StopIteration

# Использование:

for i in Fibonacci(100): print i,

```

Простые генераторы

Разработчики языка не остановились на итераторах. Как оказалось, в интерпретаторе Python достаточно просто реализовать **простые генераторы**. Под этим термином фактически понимается специальный объект, вычисления в котором продолжаются до выработки очередного значения, а затем приостанавливаются до возникновения необходимости в выдаче следующего значения. Простой генератор формируется функцией-генератором, которая синтаксически похожа на обычную функцию, но использует специальный оператор `yield` для выдачи следующего значения. При вызове такая функция ничего не вычисляет, а создает объект с интерфейсом итератора для получения значений. Другими словами, если функция должна возвращать последовательность, из нее довольно просто сделать генератор, который будет функционально эквивалентной "ленивой" реализацией. **Ленивыми** называются вычисления, которые откладываются до самого последнего момента, когда получаемое в результате значение сразу используется в другом вычислении.

Для примера с последовательностью Фибоначчи можно построить такой вот генератор:

```

def Fib(N): a, b =
0, 1

for i in xrange(N): yield a
a, b = b, a + b

```

Использовать его не сложнее, чем любой другой итератор:

```

for i in Fib(100): print
i,

```

Однако следует заметить, что программа в значительной степени упростилась.

Генераторное выражение

В Python 2.4 по аналогии со списковым включением появилось **генераторное выражение**. По синтаксису оно аналогично списковому, но вместо квадратных скобок используются круглые. Списковое включение порождает список, а, значит, можно ненароком занять очень много памяти. Генератор же, получающийся в результате применения генераторного выражения, списка не создает, он вычисляет каждое следующее значение строго по требованию (при вызове метода `next()`).

В следующем примере можно прочесть из файла строки, в которых производятся некоторые замены:

```
for line in (l.replace("- ", " - ") for l in open("input.dat")): print line
```

Ничто не мешает использовать итераторы и для записи в файл:

```
open("output.dat", "w").writelines(  
l.replace("- ", " - ") for l in open("input.dat"))
```

Здесь для генераторного выражения не потребовалось дополнительных скобок, так как оно расположено внутри скобок вызова функции.

Карринг

Библиотека Xoltar toolkit (автор Bryn Keller) включает модуль functional, который позволяет упростить использование возможностей функционального программирования. Модуль functional применяет "чистый" Python. Библиотеку можно найти по [этому адресу](#).

При **карринге** (частичном применении) функции создается новая функция, задавая некоторые аргументы исходной. Следующий пример иллюстрирует частичное применение вычитания:

```
from functional import curry def  
subtract(x, y):  
    return x - y  
  
print subtract(3, 2)  
  
subtract_from_3 = curry(subtract, 3) print  
subtract_from_3(2)  
print curry(subtract, 3)(2)
```

Во всех трех случаях будет выведено 1. В следующем примере получается новая функция, подставляя второй аргумент. Вместо другого аргумента вставляется специальное значение Blank:

```
from functional import curry, Blank def  
subtract(x, y):  
    return x - y  
  
print subtract(3, 2)  
  
subtract_2 = curry(subtract, Blank, 2) print  
subtract_2(3)  
  
print curry(subtract, Blank, 2)(3)
```

Рассмотрим самый примитивный случай, каррируем функцию `sum_x_y`:

```
sum_x_y_carry = lambda x: lambda y: sum_x_y(x, y)
```

```
print(sum_x_y_carry(5)(12)) # 17
```

Рассмотрим другой вариант:

```
sum_with_12 = sum_x_y_carry(12)
print(sum_with_12(1), sum_with_12(12)) # 13 24
sum_with_5 = sum_x_y_carry(5)
print(sum_with_12(10), sum_with_12(17)) # 22 29
```

Сделаем универсальную функцию для каррирования функций с двумя доводами:

```
curry_2 = lambda fn: lambda x: lambda y: fn(x, y)
```

И применим ее к применяемой в реальных планах функции map:

```
curry_map_2 = curry_2(map)
```

```
@curry_map_2
def twice_or_increase(n):
    if n % 2 == 0:
        n = 1
    if n % 3:
        n *= 2
    return n

print(*twice_or_increase(range(10))) # 2 2 3 3 10 10 14 14 9 9
print(*twice_or_increase(range(30))) # 2 2 3 3 10 10 14 14 9 9 22 22 26 26 15 15 34 34
38...
```

Но не все функции принимают 2 довода, следовательно сделаем функцию `curry_n`, применяя partial, замыкания и немного рекурсии:

```
from functools import partial

def curry_n(fn, n):
    def aux(x, n=None, args=None): # вспомогательная функция
        args = args + [x] # добавим довод в список всех доводов
        return partial(aux, n=n - 1, args=args) if n > 1 else fn(*args) # вернем функцию с
    одним доводом, сделанную из aux либо вызовем первоначальную с полученными
    доводами
    return partial(aux, n=n, args=[])
```

И в следующий раз применим к map, но теснее с 3 доводами:

```
curry_3_map = curry_n(map, 3)
```

И сделаем функцию для сложения элементов списка с элементами списка 1..10:

```
sum_arrays = curry_3_map(lambda x, y: x + y)
sum_with_range_10 = sum_arrays(range(10))
print(*sum_with_range_10(range(100, 0, -10))) # 100 91 82 73 64 55 46 37 28 19
print(*sum_with_range_10(range(10))) # 0 2 4 6 8 10 12 14 16 18
```

Так как `curry_2` — это частный случай `curry_n`, то дозволено сделать:

```
curry_2 = partial(curry_n, n=2)
```

И для примера применим его к filter:

```
curry_filter = curry_2(filter)
only_odd = curry_filter(lambda n: n % 2)
```

```
print(*only_odd(range(10))) # 1 3 5 7 9
print(*only_odd(range(-10, 0, 1))) # -9 -7 -5 -3 -1
```

Pattern matching

Способ обзора списков либо других конструкций данных на присутствие в них заданных примеров.

Рассмотрим задачу — написать функцию, которая:

- если принимает список чисел, возвращает их произведение;
- если принимает список строк, возвращает одну огромную объединенную строку.

Сделаем вспомогательный exception и функцию для его «бросания», которую будем применять, когда сравнение не проходит:

```
class NotMatch(Exception):
    """Not match"""

def not_match(x):
    raise NotMatch(x)
```

И функцию, которая делает проверку и возвращает объект, либо бросает exception:

```
match = lambda check, obj: obj if check(obj) else not_match(obj)
match_curry = curry_n(match, 2)
```

Сделать проверку типа:

```
instance_of = lambda type_: match_curry(lambda obj: isinstance(obj, type_))
Тогда для int:
is_int = instance_of(int)
print(is_int(2)) # 2
try:
    is_int('str')
except NotMatch:
    print('not int') # not int
```

Сделаем проверку типа для списка, проверяя его каждый элемент:

```
is_array_of = lambda matcher: match_curry(lambda obj: all(map(matcher, obj)))
```

Итогда для int:

```
is_array_of_int = is_array_of(is_int)
print(is_array_of_int([1, 2, 3])) # 1 2 3
try:
    is_array_of_int('str')
except NotMatch:
    print('not int') # not int
```

Подобно для str:

```
is_str = instance_of(str)
is_array_of_str = is_array_of(is_str)
```

Также добавим функцию, возвращающую свой довод, идемпотентную =)

```
identity = lambda x: x
```

```
print(identity(10)) # 10
print(identity(20)) # 20
```

И проверку на пустой список:

```
is_blank = match_curry(lambda xs: len(xs) == 0)
print(is_blank([])) # []
try:
    is_blank([1, 2, 3])
except NotMatch:
    print('not blank') # not blank
```

Рассмотрим функцию для распределения списка на 1-й элемент и остаток с использованием к ним «проверок»:

```
def hd_tl(match_x, match_xs, arr):
    x, *xs = arr
    return match_x(x), match_xs(xs)
```

```
hd_tl_partial = lambda match_x, match_xs: partial(hd_tl, match_x, match_xs)
```

Рассмотрим самый простой пример с identity:

```
hd_tl_identity = hd_tl_partial(identity, identity)
print(hd_tl_identity(range(5))) # 0 [1, 2, 3, 4]
```

Исчислами:

```
hd_tl_ints = hd_tl_partial(is_int, is_array_of_int)
print(hd_tl_ints(range(2, 6))) # 2 [3, 4, 5]
try:
    hd_tl_ints(['str', 1, 2])
except NotMatch:
    print('not ints') # not ints
```

Рассмотрим функцию, которая будет перебирать все проверки.

```
def pattern_match(patterns, args):
    for pattern, fnc in patterns:
        try:
            return fnc(pattern(args))
        except NotMatch:
            continue
    raise NotMatch(args)
```

```
pattern_match_curry = curry_n(pattern_match, 2)
```

Но она неудобна в применении и требует целый мир скобок, поэтому функция будет выглядеть так:

```
sum_or_multiply = pattern_match_curry((
    (hd_tl_partial(identity, is_blank), lambda arr: arr[0]), # x::[] -> x
    (hd_tl_ints, lambda arr: arr[0] * sum_or_multiply(arr[1])), # x::xs -> x *
sum_or_multiply(xs) где type(x) == int
    (hd_tl_partial(is_str, is_array_of_str), lambda arr: arr[0] sum_or_multiply(arr[1])), #
x::xs -> x sum_or_multiply(xs) где type(x) == str
))
```

Проверим ее в действии:

```
print(sum_or_multiply(range(1, 10))) # 362880  
print(sum_or_multiply(['a', 'b', 'c'])) # abc
```

2. ОБОРУДОВАНИЕ

ПЭВМ IBM PC, SVGA монитор с разрешением не менее 800*600 пикселей; клавиатура; мышь, ОС Linux/Windows, язык программирования Python, среда Geany

3 . ПОДГОТОВКА К РАБОТЕ

1. Изучить теоретическую часть.

4 . ПРОГРАММА РАБОТЫ

1. Отладить примеры, приведенные в общих положениях данной лабораторной работы.
2. Результаты работы программ показать преподавателю.
3. Оформить отчет.
4. Защитить лабораторную работу перед преподавателем.

5. СОДЕРЖАНИЕ ОТЧЕТА

1. Номер и название лабораторной работы
2. Цель и задачи
3. Текст программы
4. Результаты и выводы по лабораторной работе

ЛАБОРАТОРНАЯ РАБОТА №6.

Функциональное программирование на языке программирования Python. Часть 2

ЦЕЛЬ И ЗАДАЧИ РАБОТЫ: Изучение основ функционального программирования на языке программирования Python для решения специальных задач.

1. ОБОРУДОВАНИЕ

ПЭВМ IBM PC, SVGA монитор с разрешением не менее 800*600 пикселей; клавиатура; мышь, ОС Linux/Windows, язык программирования Python, среда Geany

2. ПОДГОТОВКА К РАБОТЕ

1. Изучить теоретическую часть (лабораторная работа №5).

3. ПРОГРАММА РАБОТЫ

1. Написать и отладить программы на Python, используя понятия и методы функционального программирования (индивидуальные задания из лабораторной работы №2).
2. Результаты работы программ показать преподавателю.
3. Оформить отчет.
4. Защитить лабораторную работу перед преподавателем.

4. СОДЕРЖАНИЕ ОТЧЕТА

1. Номер и название лабораторной работы
2. Цель и задачи
3. Текст программы
4. Результаты и выводы по лабораторной работе

ЛАБОРАТОРНАЯ РАБОТА №7.

Работа с директориями и файлами в Python

ЦЕЛЬ И ЗАДАЧИ РАБОТЫ: Изучение основ работы с директориями и файлами на языке программирования Python.

3. ОБЩИЕ ПОЛОЖЕНИЯ

Как открыть файл.

Открыть файл можно с помощью функции open:

```
open(name[, mode[, buffering]])
```

Функция возвращает файловый объект. Обязателен только первый аргумент. Если остальные параметры отсутствуют, файл будет доступен на чтение. Таблица режимов (mode) функции open:

```
'r' – чтение.  
'w' – запись.  
'a' – добавление.  
'b' – бинарный режим.  
'+' – чтение/запись.
```

Режим '+' может быть добавлен к остальным режимам. По умолчанию питон открывает файлы в текстовом режиме. Для открытия файла в бинарном режиме на чтение можно добавить 'rb'. Третий параметр устанавливает размер буферизации при работе с файлом. По умолчанию он выключен, и чтение/запись идет напрямую с диска на диск. Для включения буфера третий параметр должен быть отличным от нуля.

Базовые файловые методы.

В питоне многие объекты являются файлами: стандартный ввод sys.stdin, стандартный вывод sys.stdout, объекты, открываемые функцией urllib.urlopen и т.д.

Запись в файл:

```
>>> f = open('my_file', 'w')  
>>> f.write('Hello, ')  
>>> f.write('World!')  
>>> f.close()
```

Чтение:

```
>>> f = open('my_file', 'r')  
>>> f.read(5)  
'Hello'  
>>> f.read()  
, World!'
```

Стандартный ввод/вывод.

В командной строке можно записать подряд несколько команд, передавая результат работы от одной команды к другой по конвейеру – или по каналу (pipe):

```
cat my_file | python test.py
```

Первая команда – cat – пишет содержимое текстового файла my_file на стандартный вывод sys.stdout. Вторая команда запускает питоновский файл, который читает стандартный ввод sys.stdin, подсчитывает в нем количество слов и выводит результат:

```
test.py:
import sys
text = sys.stdin.read()
words = text.split()
wordcount = len(words)
print 'Wordcount:', wordcount
```

Канал – или пайп (pipe) – это конструкция, объединяющая стандартный вывод со стандартным вводом и позволяющая обмениваться данными между двумя командами.

Произвольный доступ.

По умолчанию метод read() читает данные последовательно по порядку, от начала и до конца файла. Для произвольного доступа к файлу есть функция seek:

```
seek(offset[, whence])
```

offset – смещение в байтах относительно начала файла;
whence – по умолчанию равен нулю, указывает на то, что смещение берется относительно начала файла.

Пример:

```
>>> f = open(r'my_file', 'w')
>>> f.write('01234567890123456789')
>>> f.seek(5)
>>> f.write('Hello, World!')
>>> f.close()
>>> f = open(r'my_file')
>>> f.read()
'01234Hello, World!89'
```

Функция tell() возвращает текущую позицию файла.

Построчная работа с файлами

Обычно мы имеем дело с текстовыми файлами. Прочитать одну строку:

```
file.readline()
```

Функция readline() без параметра читает всю строку, наличие параметра указывает функции максимальное число символов строки, которое будет прочитано. Прочитать все строки и вернуть список строк:

```
file.readlines()
```

Записать строки в файл:

```
file.writelines()
```

Пример. Прочитать файл и записать его содержимое в другой файл:

```
f = open(r'my_file')
lines = f.readlines()
f.close()
lines[0] = "This is a my_file2 \n" # изменяем 1-ю строку
f = open(r'my_file2', 'w')
f.writelines(lines)
f.close()
```

Заккрытие файла

Для закрытия файла есть метод `close()`. Обычно файл закрывается сам после того, как вы выходите из программы, но файлы нужно закрывать вручную по нескольким причинам.

1. Питон может буферизировать запись в файл ваших данных, что может привести к неожиданным эффектам и возникновению ошибок.

2. У операционной системы есть ограничение на число одновременно открытых файлов.

3. При доступе к файлу из разных мест одновременно и на чтение, и на запись необходимо синхронизировать файловые операции. Буферизация записи может привести к тому, что запись уже произошла, а данных в файле еще нет.

Для полной уверенности в закрытии файла можно использовать блок `try/finally`:

```
try:
    # Тут идет запись в файл
finally:
    file.close()
```

Можно также использовать менеджер контекста, который в любом случае закроет файл:

```
with open("my_file") as somefile:
    do_something(somefile)
```

Если вы все же не хотите закрывать файл, то синхронизировать многопользовательский доступ к файлу на чтение/запись можно с помощью функции `flush()`, которая актуализирует все операции записи на диск. При этом возможна блокировка файла на чтение.

Итерация

Итерация по файлу является базовой операцией и имеет множество вариантов. Использование функции `read()` для байтового чтения:

```
f = open(filename)
while True:
    char = f.read(1)
    if not char: break
    process(char)
f.close()
```

Построчное чтение текстовых файлов и функция `readline()`:

```
f = open(filename)
while True:
    line = f.readline()
    if not line: break
    process(line)
f.close()
```

Файл сам может выступать в роли итератора:

```
for line in open(filename):
    process(line)
```

Pickling

Практически любой тип объекта может быть сохранен на диске в любой момент его жизни, а позже прочитан с диска. Для этого есть модуль `pickle`:

```
import pickle
t1 = [1, 2, 3]
s = pickle.dumps(t1)
t2 = pickle.loads(s)
print t2
[1, 2, 3]
```

Здесь есть небольшой нюанс: `t1` и `t2` будут двумя разными объектами, хотя и идентичными.

Бинарные файлы

Стандартный модуль `struct` позволяет преобразовывать объекты в структуры C в виде строк в бинарном формате и обратно. Данные в строке располагаются в соответствии со строкой формата. Эти возможности могут быть использованы для чтения и сохранения в двоичном формате.

Функции этого модуля:

```
pack(format, value1, value2 ...)
```

Возвращает строку, содержащую значения value1 ..., упакованные в соответствии с форматом. Количество и тип аргументов должны соответствовать значениям, которые требует строка формата format.

```
unpack(format, string)
```

Распаковывает строку string в соответствии с форматом format и возвращает кортеж объектов.

```
calcsize(format)
```

Возвращает размер структуры (т.е. длину строки), соответствующей формату format.

Таблица. Таблица основных форматов

Format C TypePython		
cchar	string of length 1	
?	Boolbool	
i	int	integer
l	long	integer
f	float	float
d	double	float
s	char[]	string

Перед символом формата может идти число, обозначающее количество повторений. Например, строка формата '4h' полностью эквивалентна строке 'hhhh'. Символы пропуска между символами формата игнорируются, однако символы пропуска между числом и символом формата не допускаются.

Число перед символом формата 's' интерпретируется как длина строки, а не число повторений. То есть '10s' обозначает строку из 10 символов, в то время как '10c' – 10 раз по одному символу.

Можно изменить порядок следования байтов вручную:

```
< - little-endian  
> - big-endian
```

В следующем примере мы упаковываем в структуру два числа – целое и float, строку из пяти символов, сохраняем в бинарный файл, а потом извлекаем из файла:

```
from struct import *  
out = open("123.bin", "wb")  
format = "if5s"  
data = pack(format, 24, 12.48, '12345')
```

```

out.write(data)
out.close()
input = open("123.bin", "rb")
data = input.read()
input.close()
format = "i5s"          # one integer
value,value2,value3 = unpack(format, data) # note the ',' in 'value,':
                        unpack apparently returns a n-uple
print value
print value2
print value3
printcalcsizes(format)

>>> 24
>>> 12.4799995422
>>> 12345
>>> 13

```

Работа с файловой системой

Стандартный модуль `os` имеет интерфейс работы с файловой системой. Каждая программа имеет текущий каталог. Функция `os.getcwd` возвращает текущий каталог:

```

import os
cwd = os.getcwd()
print cwd

```

Проверить наличие файла в текущем каталоге:

```

os.path.exists('my_file')

```

Вывести список файлов и подкаталогов для данного каталога:

```

os.listdir(path)

```

Следующий пример рекурсивно выводит список всех файлов и подкаталогов для данного каталога:

```

import os
def walk(dir):
    for name in os.listdir(dir):
        path = os.path.join(dir, name)
        if os.path.isfile(path):
            print path
        else:
            walk(path)
            walk(path)

```

В следующем примере мы получим статистическую информацию о текущем каталоге: общий размер каталога в байтах, число файлов, число подкаталогов. Стандартная функция `os.path.walk` имеет три параметра: каталог, пользовательская функция, список для подсчета:

```

importos, sys
defgetlocaldata(sms,dr,flst):
for f in flst:
fullf = os.path.join(dr,f)
ifos.path.islink(fullf): continue # don't count linked files
ifos.path.isfile(fullf):
sms[0] += os.path.getsize(fullf)
sms[1] += 1
else:
sms[2] += 1
defdtstat(dtroot):
sums = [0,0,1] # 0 bytes, 0 files, 1 directory so far
os.path.walk(dtroot,getlocaldata,sums)
return sums

report = dtstat('.')
printreport

```

В следующем примере сделана интерпретация системной утилиты `grep`. В текущем каталоге будут найдены файлы с питоновским расширением, в которых будет найдена поисковая строка `'importos'`:

```

importos, sys, fnmatch

mask = '*.py'
pattern = 'import os'

def walk(arg,dir,files):
for file in files:
iffnmatch.fnmatch(file,mask):
name = os.path.join(dir,file)
try:
data = open(name,'rb').read()
ifdata.find(pattern) != -1:
print name
except:
pass
os.path.walk('.',walk,[])

```

Для работы с файлами в основном применяется модуль `os`. Для перемещения и копирования файлов используется модуль `shutil`, для сравнения файлов - `filecmp`, для поиска по образцу - `glob`. Импортируем `os`:

Обработка пути к файлу

```

os.path.join(r'c:\home', 'index.html') # соединение пути

```

```

# возвращает r'c:\home\index.html'

os.path.split(r'c:\home\index.html') # разделение пути
# возвращает (r'c:\home', 'index.html')

```

Как узнать, существует ли данный путь?

```
path = 'index.html'
```

```
if os.path.exists(path):  
    print path, ' exists'
```

Как определить, данный путь - это файл или директория?

```
if os.path.isfile(path):
```

```
    print path, ' is file'  
elif os.path.isdir(path):  
    print path, ' is directory'
```

Получаем свойства файла.

```
from datetime import datetime # для преобразования даты в приемлемый формат
```

```
size = os.path.getsize(path) # размер файла в байтах  
ksize = size//1024           # размер в килобайтах  
atime = os.path.getatime(path) # дата последнего доступа в секундах с начала эпохи  
mtime = os.path.getmtime(path) # дата последней модификации в секундах с начала эпохи  
  
print 'Size: ', ksize, ' KB'  
print 'Last access date: ', datetime.fromtimestamp(atime)  
print 'Last modification date: ', datetime.fromtimestamp(mtime)
```

Стандартные операции с файлами

```
import shutil
```

```
os.rename('index.html', 'index2.html') # переименовать  
shutil.copy('index2.html', 'index.html') # копировать  
shutil.move('index2.html', 'index3.html') # переместить  
os.remove('index3.html') # удалить
```

Сравнение двух файлов

```
import filecmp
```

```
eq = filecmp.cmp('index.html', 'index3.html') # сравнить файлы  
if eq:  
    print 'equal'
```

```
else:
    print 'notequal'
```

Обработка всех файлов в директории

```
dir = 'c:\home'
```

```
names = os.listdir(dir)    # список файлов и поддиректорий в данной директории
for name in names:
    fullname = os.path.join(dir, name) # получаем полное имя
    if os.path.isfile(fullname):        # если это файл...
        print fullname                # делаем что-нибудь с ним
```

Обработка всех файлов в директории рекурсивно (включая все поддиректории)

```
dir = 'c:\home'
```

```
for root, dirs, files in os.walk(dir): # пройти по директории рекурсивно
    for name in files:
        fullname = os.path.join(root, name) # получаем полное имя файла
        print fullname                      # делаем что-нибудь с ним
```

Здесь `os.walk` - это итератор, который проходит по директориям рекурсивно, и возвращает для каждой директории кортеж из трёх значений:

`root` - имя текущей директории

`dirs` - имена поддиректорий в текущей директории

`files` - имена файлов в текущей директории

Получение списка файлов, совпадающих с образцом

```
import glob
```

```
names = glob.glob('c:\home\*')    # все файлы и поддиректории в "c:\home"
names = glob.glob('c:\home\*.txt') # только с расширением ".txt"

for name in names:
    if os.path.isfile(name): # если это файл (а не директория)
        print name          # делаем что-нибудь с ним
```

Примеры работы с файлами в Python

Откроем файл на запись.

```
>>> f = open('C:/test/test.txt', 'w')
```

Запишем в него несколько строк с переносом на другую строку `"\r\n"`.

```
>>> f.write('string1 \r\n')
```

10

```
>>>f.write('string2')
```

7

Закроем файл

```
>>>f.close()
```

Пример с чтением файла по количеству байт, и "прочитать все"

```
>>> f = open('C:/test/test.txt', 'r')
```

```
>>>f.read(7)
```

```
'string1'
```

```
>>>f.read()
```

```
'\n\nstring2'
```

```
>>>f.close()
```

Пример с редактированием файла по позиции байт

```
>>> f = open('C:/test/test.txt', 'w')
```

```
>>>f.write('abcdef')
```

6

```
>>>f.seek(3)
```

3

```
>>>f.write('CD')
```

2

```
>>>f.close()
```

```
>>> f = open('C:/test/test.txt', 'r')
```

```
>>>f.read()
```

```
'abcCDf'
```

Еще пример чтения из файла

```
>>> f = open('C:/test/test.txt', 'w')
```

```
>>>f.write('string \r\n')
```

9

```
>>>f.write('string \r\n')
```

9

```
>>>f.write('string \r\n')
```

9

```
>>>f.close()
```

```
>>> f = open('C:/test/test.txt', 'r')
```

```
>>>f.readline()
```

```
'string \n'
```

```
>>>f.readlines()
```

```
['\n', 'string \n', '\n', 'string \n', '\n']
```

```
>>> f.close()
```

Замена определенной строки на значение

```
>>> f = open('C:/test/test.txt', 'r')
```

```
>>> strings = f.readlines()
```

```
>>>f.close()
```

```
>>> strings[0] = 'STRING'
```

```
>>> f = open('C:/test/test.txt', 'w')
```

```
>>>f.writelines(strings)
```

```
>>>f.close()
```

4. ОБОРУДОВАНИЕ

ПЭВМ IBM PC, SVGA монитор с разрешением не менее 800*600 пикселей; клавиатура; мышь, ОС Linux/Windows, язык программирования Python, среда Geany

3 . ПОДГОТОВКА К РАБОТЕ

1. Изучить теоретическую часть.

4 . ПРОГРАММА РАБОТЫ

1. Отладить примеры, приведенные в общих положениях данной лабораторной работы.
2. Написать и отладить программу, соответствующему индивидуальному заданию (приложение А, частьА)
3. Результаты работы программ показать преподавателю.
4. Оформить отчет.
5. Защитить лабораторную работу перед преподавателем.

5. СОДЕРЖАНИЕ ОТЧЕТА

1. Номер и название лабораторной работы
2. Цель и задачи
3. Тексты программ и результаты выполнения.
4. Результаты и выводы по лабораторной работе

Приложение А

ЧАСТЬ А

Вариант 1.

Ввести и сохранить в файле данные следующей структуры: ф.,и.,о. студента, номер зачетной книжки, информация об аттестации ($>30\%$ - аттестован) по предмету (5 предметов). Организовать просмотр исходных данных и вывести список студентов, отсортированный по убыванию количества неаттестаций (методом вставки), с указанием номера зачетной книжки и результатов аттестации только по тем предметам, по которым процент аттестации $<30\%$.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 2.

Ввести и сохранить в файле данные следующей структуры: ф.,и.,о. студента, номер группы, результаты сдачи сессии (5 экзаменов). Организовать просмотр исходных данных и вывести список студентов, имеющих средний балл за сессию > 4.0 (сортировка по убыванию метод Шелла), с указанием номера группы и результатов по каждому экзамену.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 3.

Ввести и сохранить в файле данные следующей структуры: ф.,и.,о. водителя, номер и марка автомашины (5-7 марок), дата техосмотра. Организовать просмотр исходных данных и вывести список владельцев автомашин определенной марки (по запросу), отсортированных по возрастанию номеров (метод пузырька), с указанием номера машины и даты техосмотра.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 4.

Ввести и сохранить в файле данные следующей структуры: марка телевизора, № квитанции, продолжительность ремонта, стоимость ремонта, дата приема. Организовать просмотр исходных данных и вывести список заказов, отсортированный по дате завершения ремонта (по возрастанию метод выборки записи с максимальным значением), с указанием стоимости заказа и даты завершения ремонта.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 5.

Ввести и сохранить в файле данные следующей структуры: ф.,и.,о. водителя, номер и марка автомашины (5-7 марок), год приобретения. Организовать просмотр исходных данных и вывести список водителей, отсортированный по году приобретения автомашины (возрастание метод Шелла), с указанием номера машины и года приобретения.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 6.

Ввести и сохранить в файле данные следующей структуры: ф.,и.,о. вкладчика, номер счета, сумма вклада, год открытия вклада. Организовать просмотр исходных данных и вывести общий список вкладчиков, отсортированный по сумме вклада (убывание метод слияния), с указанием суммы вклада и года открытия.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 7.

Ввести и сохранить в файле данные следующей структуры: ф.,и.,о. вкладчика, номер счета, сумма вклада, год открытия вклада. Организовать просмотр исходных данных и вывести общий список вкладчиков, отсортированный по году открытия вклада (возрастание метод вставки), с указанием суммы вклада и года открытия.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 8.

Ввести и сохранить в файле данные следующей структуры: ф.,и.,о. вкладчика, номер счета, сумма вклада, вид вклада (простой, срочный, до востребования). Организовать просмотр исходных данных и вывести список вкладчиков (по заданному виду вклада), отсортированный по сумме вклада (убывание метод вставки), с указанием суммы вклада и номера счета.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 8.

Ввести и сохранить в файле данные следующей структуры: ф.,и.,о. абитуриента, средний балл аттестата, номер экзаменационного листа, номер школы (не менее 5 школ). Организовать просмотр исходных данных и вывести список абитуриентов данной школы (по выбору), отсортированных по среднему баллу аттестата (убывание метод пузырька), с указанием среднего балла и номера экзаменационного листа.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 9.

Ввести и сохранить в файле данные следующей структуры: ф.,и.,о. абитуриента, средний балл аттестата, оценки, полученные на вступительных экзаменах, место жительства. Организовать просмотр исходных данных и вывести список абитуриентов, отсортированный по сумме баллов, полученных на вступительных экзаменах (убывание, метод Шелла), с указанием среднего балла и результатов экзаменов.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 10.

Ввести и сохранить в файле данные следующей структуры: ф.,и.,о. абитуриента, средний балл аттестата, оценки, полученные на вступительных экзаменах, место жительства. Организовать просмотр исходных данных и вывести список абитуриентов, не прошедших по конкурсу, отсортированный по сумме баллов, полученных на вступительных экзаменах (убывание, метод вставки), с указанием среднего балла и результатов экзаменов.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 11.

Ввести и сохранить в файле данные следующей структуры: ф.,и.,о. пассажира, пункт назначения (не менее 5 пунктов); пункт отправления (не менее 5 пунктов), вес багажа. Организовать просмотр исходных данных и вывести список пассажиров,

следующих в заданный пункт назначения (по выбору), отсортированный по весу багажа (метод обмена по возрастанию), с указанием пункта отправления и веса багажа.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 12.

Ввести и сохранить в файле данные следующей структуры: ф.,и.,о. пассажира, пункт назначения (не менее 5 пунктов); стоимость билета; номер поезда. Организовать просмотр исходных данных и вывести список всех пассажиров, отсортированный по стоимости билета (возрастание метод вставки), с указанием пункта назначения и номера поезда.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 13.

Ввести и сохранить в файле данные следующей структуры: ф.,и.,о. абитуриента, средний балл аттестата, номер экзаменационного листа, наличие льгот, сдача пробных экзаменов (2 экзамена). Организовать просмотр исходных данных и вывести список абитуриентов, отсортированный по номерам экзаменационных листов (возрастание метод слияния), с указанием номера экзаменационного листа, среднего балла аттестата и результатов по каждому экзамену.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 14.

Ввести и сохранить в файле данные следующей структуры: ф.,и.,о. абитуриента, средний балл аттестата, номер экзаменационного листа, наличие льгот, сдача пробных экзаменов (2 экзамена). Организовать просмотр исходных данных и вывести список абитуриентов-льготников, отсортированный по среднему баллу аттестата (убывание метод слияния), с указанием номера экзаменационного листа, среднего балла аттестата и результатов по каждому экзамену.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 15.

Ввести и сохранить в файле данные следующей структуры: ф.,и.,о. студента, место отдыха (не менее 5 мест), продолжительность отдыха, стоимость путевки. Организовать просмотр исходных данных и вывести список путевок в данное место отдыха (по выбору), отсортированный по стоимости (возрастание метод вставки), с указанием продолжительности отдыха и стоимости.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 16.

Ввести и сохранить в файле данные следующей структуры: место отдыха (не менее 5 мест), дата заезда, продолжительность отдыха, стоимость путевки. Организовать просмотр исходных данных и вывести список путевок, отсортированный по продолжительности отдыха (убывание метод пузырька), с указанием даты заезда, продолжительности и стоимости.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 17.

Ввести и сохранить в файле данные следующей структуры: ф.,и.,о. читателя, номер читательского билета, количество книг на руках, наличие нарушения при пользовании

библиотекой. Организовать просмотр исходных данных и вывести список читателей, отсортированный по количеству книг на руках (убывание метод Шелла), с указанием номера читательского билета, количества книг и нарушений.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 18.

Ввести и сохранить в файле данные следующей структуры: ф.,и.,о. читателя, номер читательского билета, количество книг на руках, количество нарушений при пользовании библиотекой. Организовать просмотр исходных данных и вывести список нарушителей, отсортированный по количеству нарушений (убывание метод шейкер-сортировки), с указанием номера читательского билета, количества книг и нарушений.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 19.

Ввести и сохранить в файле данные следующей структуры: туристическая фирма, место отдыха, количество дней, стоимость путевки. Организовать просмотр исходных данных и вывести список фирм, предоставляющих путевки в данное место отдыха (по выбору), отсортированный по стоимости путевки (возрастание метод вставки), с указанием количества дней и стоимости.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 20.

Ввести и сохранить в файле данные следующей структуры: название отеля, стоимость номера по количеству комнат (1, 2, 3), количество номеров в отеле с 1, 2, 3 комнатами, район города. Организовать просмотр исходных данных и вывести список отелей, отсортированный по убыванию стоимости номера (метод пузырька) для одно-, двух-, трехкомнатных номеров (по выбору), с указанием стоимости и района города.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 21.

Ввести и сохранить в файле данные следующей структуры: название отеля, стоимость номера по количеству комнат (1, 2, 3), количество номеров в отеле с 1, 2, 3 комнатами, район города. Организовать просмотр исходных данных и вывести список отелей в данном районе (по выбору), отсортированный по возрастанию стоимости номера (метод вставки) для одно-, двух-, трехкомнатных номеров, с указанием стоимости и количества номеров.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 22.

Ввести и сохранить в файле данные следующей структуры: наименование товара, фирма изготовитель, цена, гарантия. Организовать просмотр исходных данных и вывести список товаров, отсортированный по гарантийному сроку (убывание метод пузырька), с указанием фирмы изготовителя и гарантии.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 23.

Ввести и сохранить в файле данные следующей структуры: наименование товара, цвет, цена, размер. Организовать просмотр исходных данных и вывести список товаров определенного наименования, отсортированный по цене (убывание метод вставки), с указанием цены, цвета и размера.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 24.

Ввести и сохранить в файле данные следующей структуры: категория товара, наименование товара, цвет, цена, размер. Организовать просмотр исходных данных и вывести список товаров определенной категории (по выбору) и размера, отсортированный по стоимости (возрастание метод пузырька), с указанием цены, цвета.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 25.

Ввести и сохранить в файле данные следующей структуры: наименование товара, фирма изготовитель, срок хранения, стоимость товара, количество на складе. Организовать просмотр исходных данных и вывести список товаров, количество на складе которых не меньше заданного, отсортированный по возрастанию методом пузырька, с указанием срока хранения и стоимости.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 26.

Ввести и сохранить в файле данные следующей структуры: наименование фирмы, наименование покупаемых (продаваемых) акций, стоимость продажи (покупки) акций. Организовать просмотр исходных данных и вывести список фирм, покупающих акции, отсортированный по убыванию стоимости методом вставки, с указанием наименования и стоимости акции.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 27.

Ввести и сохранить в файле данные следующей структуры: наименование фирмы, адрес, наименование покупаемых (продаваемых) акций, стоимость продажи (покупки) акций. Организовать просмотр исходных данных и вывести список фирм, продающих данные акции (по выбору), отсортированный по возрастанию стоимости методом шейкер-сортировки, с указанием стоимости акции и адреса фирмы.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 28.

Ввести и сохранить в файле данные следующей структуры: наименование фирмы, количество комнат в продаваемой квартире, стоимость, тип дома, район. Организовать просмотр исходных данных и вывести список всех квартир, расположенных в определенном районе (по выбору), отсортированный по стоимости (убывание метод Шелла), с указанием количества комнат и стоимости.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 29.

Ввести и сохранить в файле данные следующей структуры: наименование фирмы, количество комнат в продаваемой квартире, стоимость, тип дома, район. Организовать просмотр исходных данных и вывести список всех квартир, отсортированный по количеству комнат (убывание метод вставки), с указанием количества комнат, типа дома и стоимости.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 30.

Ввести и сохранить в файле данные следующей структуры: наименование фирмы, место расположения продаваемого жилья (город, улица), тип жилья (квартира, частный дом), площадь жилья. Организовать просмотр исходных данных и вывести список продаваемого жилья данного типа (по выбору), отсортированный по площади (убывание метод пузырька), с указанием места расположения, площади.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

ЧАСТЬ Б

Вариант 31.

Ввести и сохранить в файле данные следующей структуры: наименование фирмы, адрес фирмы, наименование товара, розничная цена, оптовая цена. Организовать просмотр исходных данных и вывести список фирм, производящих оптовую продажу товара определенного наименования (по выбору), отсортированных по возрастанию цены (метод пузырька), с указанием адреса и цены.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 32.

Ввести и сохранить в файле данные следующей структуры: наименование банка, наименование обмениваемых валют, курс покупки – продажи валюты. Организовать просмотр исходных данных и вывести список банков, отсортированный по убыванию курса продажи валюты (метод слияния), с указанием наименования валюты и курса.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 33.

Ввести и сохранить в файле данные следующей структуры: наименование фирмы, наименование товара, стоимость товара, размер скидки при покупке более 10 единиц товара, стоимость перевозки, количество товара. Организовать просмотр исходных данных и вывести список фирм, предоставляющих скидку более заданной (по выбору), отсортированный по стоимости товара (убывание метод вставки), с указанием наименования товара, стоимости товара, размера скидки, стоимости с учетом скидки и количества.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 34.

Ввести и сохранить в файле данные следующей структуры: наименование фирмы, наименование товара, стоимость единицы товара, стоимость перевозки, количество товара. Организовать просмотр исходных данных и вывести список фирм, отсортированный по стоимости затрат на приобретение и перевозку товара (возрастание метод пузырька), с указанием наименования товара, количества и общих затрат.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 35.

Ввести и сохранить в файле данные следующей структуры: наименование фирмы, марка компьютера, стоимость компьютера, гарантийный срок обслуживания, устанавливаемая система при продаже (MS-DOS, Windows). Организовать просмотр исходных данных и вывести список всех фирм, отсортированный по возрастанию стоимости (метод Шелла) и имеющих гарантию более 1 года, с указанием марки, стоимости и срока гарантии.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 36.

Ввести и сохранить в файле данные следующей структуры: номер рейса, пункт отправления, пункт назначения, время отправления, продолжительность полета, стоимость билета. Организовать просмотр исходных данных и вывести список самолетов данного направления (по выбору), отсортированный по возрастанию продолжительности полета (метод пузырька), с указанием номера рейса, времени отправления и стоимости билета.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 37.

Ввести и сохранить в файле данные следующей структуры: ф.,и.,о. водителя, номер и марка автомашины (5-7 марок), год приобретения, количество нарушений. Организовать просмотр исходных данных и вывести список нарушителей (сортировка по количеству нарушений по возрастанию метод Шелла), с указанием ф.,и.,о. водителя, номера машины и количества нарушений.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 38.

Ввести и сохранить в файле данные следующей структуры: фамилия лечащего врача, фамилия пациента, возраст, заболевание, количество обращений к врачам. Организовать просмотр исходных данных и вывести список пациентов, отсортированный по количеству обращений к врачам (по убыванию метод вставки), с указанием возраста, заболеваний, количества обращений.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 39.

Ввести и сохранить в файле данные следующей структуры: название фильма, режиссер, цена, продолжительность фильма, количество проданных кассет. Организовать просмотр исходных данных и вывести список фильмов, отсортированный с учетом популярности (по убыванию метод пузырька), с указанием режиссера, цены, количества проданных кассет.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 40.

Ввести и сохранить в файле данные следующей структуры: номер заказа, тип телевизора, количество комплектующих деталей, фамилия мастера (4 мастера), срок

исполнения. Организовать просмотр исходных данных и вывести список заказов, отсортированный по количеству необходимых комплектующих деталей (по возрастанию метод Шелла), с указанием фамилии мастера, срока исполнения.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 41.

Ввести и сохранить в файле данные следующей структуры: вид животного, кличка, возраст, рацион, номер занимаемого им помещения, фамилия работника зоопарка. Организовать просмотр исходных данных и вывести список животных данного вида (по выбору), отсортированный по возрасту (по убыванию метод обмена), с указанием возраста, рациона, номера занимаемого помещения.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 42.

Ввести и сохранить в файле данные следующей структуры: вид животного, кличка, возраст, рацион, номер занимаемого им помещения, фамилия работника зоопарка. Организовать просмотр исходных данных и вывести список животных, отсортированный по номеру помещения (по возрастанию метод шейкер-сортировки), с указанием вида, клички, рациона, номера занимаемого помещения.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 43.

Ввести и сохранить в файле данные следующей структуры: ф.,и.,о. абитуриента, дата рождения, наличие золотой или серебряной медали, средний балл аттестата, выбранная специальность. Организовать просмотр исходных данных и вывести список абитуриентов (с медалями и без медалей), отсортированный по среднему баллу аттестата (по убыванию метод вставки), с указанием даты рождения, среднего балла и выбранной специальности.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 44.

Ввести и сохранить в файле данные следующей структуры: код товара, наименование товара, количество товара на складе, наименование магазина (не менее 4), количество товара в магазине, стоимость товара. Организовать просмотр исходных данных и вывести список товара определенного наименования (по выбору), отсортированный по стоимости (по убыванию метод слияния), с указанием кода товара, стоимости, количества товара на складе и в магазине.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 45.

Ввести и сохранить в файле данные следующей структуры: личный номер, ФИО, образование (высшее, среднее техническое, среднее), отдел, должность, стаж. Организовать просмотр исходных данных и вывести список сотрудников, отсортированный по стажу работы (по убыванию метод Шелла), с указанием личного номера, ФИО, образования, стажа.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 46.

Ввести и сохранить в файле данные следующей структуры: личный номер, ФИО, образование (высшее, среднее техническое, среднее), должность, оклад. Организовать просмотр исходных данных и вывести список сотрудников по уровню образования (по выбору), отсортированный по размеру оклада (по убыванию метод обмена), с указанием

личного номера, ФИО, должности.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 47.

Ввести и сохранить в файле данные следующей структуры: № квитанции, наименование товара, дата поступления товара, срок, на который сдан товар, стоимость товара. Организовать просмотр исходных данных и вывести список товаров, отсортированный по дате поступления (по возрастанию метод пузырька) с указанием № квитанции, наименования товара и даты поступления.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 48.

Ввести и сохранить в файле данные следующей структуры: номер квитанции, Ф.И.О. клиента, марка автомобиля, № автомобиля, вид неисправности, стоимость ремонта. Организовать просмотр исходных данных и вывести список автомобилей, отсортированный по стоимости ремонта (по возрастанию метод обмена), с указанием номера квитанции, Ф.И.О. клиента, вида неисправности, стоимости ремонта.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 49.

Ввести и сохранить в файле данные следующей структуры: наименование товара, страна-производитель, цена товара, количество товара, фирма-поставщик, номер отдела универмага. Организовать просмотр исходных данных и вывести список товаров, отсортированный по номеру отдела (по убыванию метод Шелла), с указанием цены товара, количества и номера отдела.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 50.

Ввести и сохранить в файле данные следующей структуры: наименование товара, страна-производитель, цена товара, количество товара, фирма-поставщик, номер отдела универмага. Организовать просмотр исходных данных и вывести список товаров, производимых в данной стране (по выбору), отсортированный по количеству (по убыванию метод пузырька), с указанием цены товара, количества и фирмы-поставщика.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 51.

Ввести и сохранить в файле данные следующей структуры: регистрационный номер, марка, номер двигателя, год выпуска, дата последнего техосмотра. Организовать просмотр исходных данных и вывести список автомобилей, отсортированный по году выпуска (по возрастанию метод пузырька), с указанием регистрационного номера, марки, года выпуска.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 52.

Ввести и сохранить в файле данные следующей структуры: тип самолета (пассажирский, грузовой), номер самолета, номер рейса, пункт отправления, пункт назначения, время отправления, время полета. Организовать просмотр исходных данных и вывести список самолетов, отсортированный по номеру рейса (по возрастанию метод Шелла), с указанием номера рейса, пункта отправления, пункта назначения, времени отправления, времени полета.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 53.

Ввести и сохранить в файле данные следующей структуры: код товара, название товара, количество на складе, страна-производитель, дата выработки, срок годности. Организовать просмотр исходных данных и вывести список товаров, отсортированный по количеству на складе (по возрастанию метод слияния) с указанием кода товара, названия товара, количества на складе.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 54.

Ввести и сохранить в файле данные следующей структуры: ФИО спортсмена, вид спорта, страна, занятое место и количество набранных баллов. Организовать просмотр исходных данных и вывести список спортсменов данной вида спорта (по выбору), отсортированный по количеству набранных баллов (по убыванию метод вставки), с указанием страны, занятого места и количества набранных баллов.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 55.

Ввести и сохранить в файле данные следующей структуры: № рейса, направление движения и время отправления автобуса, стоимость билета, ФИО водителя. Организовать просмотр исходных данных и вывести список автобусов, отсортированный по времени отправления автобуса (по возрастанию метод выборки записи с максимальным значением) с указанием № рейса, направления движения, времени отправления.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 56.

Ввести и сохранить в файле данные следующей структуры: тип поезда (пассажирский, скорый, грузовой), номер поезда, пункт отправления, пункт назначения, время отправления, стоимость билета. Организовать просмотр исходных данных и вывести список поездов, в зависимости от типа поезда (по выбору), отсортированный по номеру поезда (по возрастанию метод обмена), с указанием номера поезда, пункта отправления, пункта назначения.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 57.

Ввести и сохранить в файле данные следующей структуры: название книги, автор, год издания, количество экземпляров, тематика. Организовать просмотр исходных данных и вывести список книг заданной тематики (по выбору), отсортированный по количеству экземпляров (по убыванию метод вставки), с указанием названия книги, автора, количества экземпляров.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 58.

Ввести и сохранить в файле данные следующей структуры: название книги, автор, год издания, количество экземпляров, тематика. Организовать просмотр исходных данных и вывести список книг, отсортированный по году издания (по возрастанию метод Шелла), с указанием названия книги, автора, года издания.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 59.

Ввести и сохранить в файле данные следующей структуры: № номера, стоимость проживания, ФИО проживающего, дата заезда и срок проживания. Организовать

просмотр исходных данных и вывести список проживающих, отсортированный по сроку проживания (по возрастанию метод слияния), с указанием занимаемого номера, даты заезда и срока проживания.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 60.

Ввести и сохранить в файле данные следующей структуры: наименование лекарства, страна производитель, стоимость, объем поставки, наименование аптеки. Организовать просмотр исходных данных и вывести список лекарств, отсортированный по объему поставки (по убыванию метод вставки), с указанием стоимости, наименования аптеки и объема поставки.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 61.

Ввести и сохранить в файле данные следующей структуры: название спектакля, дата, время начала спектакля, расположение места (партер, амфитеатр, балкон), № ряда и места, стоимость билета. Организовать просмотр исходных данных и вывести программу спектаклей, отсортированную по дате (по возрастанию методом пузырька), с указанием даты, стоимости билета, времени начала спектакля.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.

Вариант 62.

Ввести и сохранить в файле данные следующей структуры: название фильма, режиссер, актеры, цена, жанр, продолжительность. Организовать просмотр исходных данных и вывести список фильмов данного жанра (по выбору) по возрастанию цены (метод обмена), с указанием названия фильма, режиссера, цены.

Ввод и вывод данных организовать в виде таблиц. Отладку программы производить на примере файла, состоящего не менее чем из 15 записей.