

Документ подписан простой электронной подписью
Информация о владельце:
ФИО: Соловьев Андрей Борисович
Должность: Директор
Дата подписания: 27.09.2023 10:42:55
Уникальный программный ключ:
с83cc511feb01f5417b9362d2700339df14aa123



МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
**ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ (ФИЛИАЛ)
ФЕДЕРАЛЬНОГО ГОСУДАРСТВЕННОГО БЮДЖЕТНОГО
ОБРАЗОВАТЕЛЬНОГО УЧРЕЖДЕНИЯ ВЫСШЕГО ОБРАЗОВАНИЯ
«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
В Г. ТАГАНРОГЕ РОСТОВСКОЙ ОБЛАСТИ
ПИ (филиал) в г. Таганроге**

ЦМК «Прикладная информатика»

Практикум

По выполнению практических работ
по дисциплине
«Основы алгоритмизации и программирования»
для специальности 09.02.07 Информационные системы и программирование

Таганрог
2023 г.

Составители: Е.С. Филонова

Практикум по выполнению практических работ по дисциплине «Основы алгоритмизации и программирования». ПИ (филиала) ДГТУ в г.Таганроге, 2023г.

В практикуме кратко изложены теоретические вопросы, необходимые для успешного выполнения практических работ, рабочее задание и контрольные вопросы для самопроверки.

Предназначено для обучающихся по специальности 09.02.07 «Информационные системы и программирование».

Ответственный за выпуск:

Председатель ЦМК: _____ О.В. Андриян

СОДЕРЖАНИЕ ПРАКТИЧЕСКИХ РАБОТ

Содержание

1	Область применения методических указаний.....	3
2	Основные задачи проведения практических работ.....	4
3	Организация и проведение практических работ.....	5
4	Практические работы.....	6
5	Индивидуальное задание.....	73
6	Основные показатели оценки результата и их критерии.....	73
7	Перечень использованных информационных ресурсов.....	74

Область применения методических указаний

В учебно-методических указаниях к практикуму по курсу «Основы алгоритмизации и программирования» изложены сведения, необходимые для успешного выполнения практических занятий по данному курсу. Описан процесс работы с инструментарием, применяемым в практических работах, представлен ряд типичных задач и подходы к их решению. Практические занятия посвящены знакомству обучающихся с текстовыми редакторами, графическими редакторами, электронными таблицами, базами данных и компьютерными сетями. Цель настоящего пособия – помочь обучающимся при выполнении практических работ, выполняемых для закрепления знаний по теоретическим основам и получения практических навыков работы на компьютерах.

Обучающийся должен знать: составлять программы для реализации методов и алгоритмов обработки различных данных; использовать технологию работы на персональной ЭВМ; использовать основные современные методы и средства разработки алгоритмов и программ; использовать основные приемы структурного программирования; использовать способы записи алгоритмов на процедурном языке программирования. Обучающийся должен уметь: методы и алгоритмы обработки различных данных, решения типовых задач; основные современные методы и средства разработки алгоритмов и программ; формальные методы, технологии и инструменты разработки программного продукта; основные приемы структурного программирования; способы записи алгоритмов на процедурном языке программирования; сферы применения рассматриваемых алгоритмов и методов. Данные учебно-методические указания предназначены для обучающихся 2 курса.

Основные задачи проведения практических работ

Требования к результатам освоения дисциплины

В результате освоения дисциплины обучающийся должен:

уметь:

1. разрабатывать схемы работы программы (блок-схемы);
2. разрабатывать программное обеспечение с помощью языков программирования;
3. осуществлять выбор метода отладки программ;
4. решать задачи тестирования и отладки программного обеспечения;
5. использовать инструментальные среды поддержки разработки, системы управления;
6. идентифицировать, анализировать и структурировать данные;

знать:

1. свойства алгоритма: конечность, определенность, результативность, массовость;
2. область определения алгоритма;
3. базовые структуры алгоритмов: линейные, разветвляющиеся, циклические;
1. базовые и динамические структуры данных и операции над ними;
2. операторы языка Python.

Формируемые компетенции:

ОК 1. Понимать сущность и социальную значимость своей будущей профессии, проявлять к ней устойчивый интерес.

ОК 2. Организовывать собственную деятельность, выбирать типовые методы и способы выполнения профессиональных задач, оценивать их эффективность и качество.

ОК 4. Осуществлять поиск и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития.

ОК 5. Осуществлять устную и письменную коммуникацию на государственном языке с учетом особенностей социального и культурного контекста.

ОК 9. Ориентироваться в условиях частой смены технологий в профессиональной деятельности.

ОК 10. Пользоваться профессиональной документацией на государственном и иностранном языках.

Организация и проведение практических работ

Практические занятия выполняются в аудитории, в компьютерном классе.

Перед выполнением работы учащийся должен отчитаться перед преподавателем за выполнение предыдущего занятия (сдать отчет).

Обучающийся должен на уровне понимания и воспроизведения предварительно усвоить необходимую для выполнения практических занятий теоретическую и информацию.

Обучающийся, получивший положительную оценку и сдавший отчет по предыдущему практическому занятию, допускается к выполнению следующему занятию.

Обучающийся, пропустивший практическое занятие по уважительной либо неуважительной причине, закрывает задолженность в процессе выполнения последующих практических занятий.

каждым обучающимся самостоятельно в полном объеме и согласно содержанию методических указаний.

Практические работы

Практическое занятие 1 Введение в язык программирования Python

Цель работы: познакомиться со средой разработки Python. Изучить основные типы данных, команды ввода и вывода данных.

Краткая теория

Python— это объектно-ориентированный, интерпретируемый, переносимый язык сверхвысокого уровня. Программирование на Python позволяет получать быстро и качественно необходимые программные модули.

В комплекте вместе с интерпретатором Python идет IDLE (интегрированная среда разработки). По своей сути она подобна интерпретатору, запущенному в интерактивном режиме с расширенным набором возможностей (подсветка синтаксиса, просмотр объектов, отладка и т.п.).

Для запуска IDLE в Windows необходимо перейти в папку Python в меню “Пуск” и найти там ярлык с именем “IDLE (Python 3.X XX-bit)”.

Для запуска редактора программы (кода) следует выполнить команду File->New File или сочетание клавиш Ctrl+N.

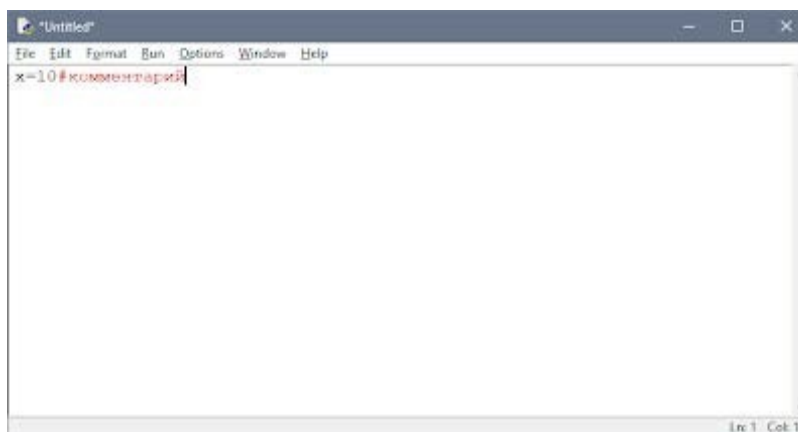
Любая Python-программа состоит из последовательности допустимых символов, записанных в определенном порядке и по определенным правилам.

Программа включает в себя:

1. комментарии;
2. команды;
3. знаки пунктуации;
4. идентификаторы;
5. ключевые слова.

Комментарии в Python обозначаются предваряющим их символом # и продолжаются до конца строки (т.е. в Python все комментарии являются однострочными), при этом не допускается использование перед

СИМВОЛОМ # КАВЫЧЕК:



Знаки пунктуации

В алфавит Python входит достаточное количество знаков пунктуации, которые используются для различных целей. Например, знаки "+" или "*" могут использоваться для сложения и умножения, а знак запятой "," - для разделения параметров функций.

Идентификаторы

Идентификаторы в Python это **имена используемые для обозначения переменной, функции, класса, модуля или другого объекта.**

Ключевые слова

Некоторые слова имеют в Python **специальное назначение и представляют собой управляющие конструкции языка.**

Ключевые слова в Python:

```
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del',  
'elif',  
'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda',  
'nonlocal',  
'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

Типы данных

1. None (неопределенное значение переменной)
2. Логические переменные (Boolean Type)

3. Числа (Numeric Type)
 1. int – целое число
 2. float – число с плавающей точкой
 3. complex – комплексное число
4. Списки (Sequence Type)
 1. list – список
 2. tuple – кортеж
 3. range – диапазон
5. Строки (Text Sequence Type)
 1. str

Ввод и вывод данных

Ввод данных осуществляется при помощи команды **input**(список ввода): `a = input()`

```
print(a)
```

В скобках функции можно указать сообщение - комментарий к вводимым данным:

```
a = input ("Введите количество: ")
```

Команда `input()` по умолчанию воспринимает входные данные как строку символов. Поэтому, чтобы ввести целочисленное значение, следует указать тип данных `int()`:

```
a = int (input())
```

Для ввода вещественных чисел применяется

```
команда a=float(input())
```

Вывод данных осуществляется при помощи команды **print**(список вывода): `a = 1`

```
b = 2
```

```
print(a)
```

```
print(a + b)
```

```
print('сумма = ', a + b)
```

Существует возможность записи команд в одну строку, разделяя их через ;. Однако не следует часто использовать такой способ, это снижает удобочитаемость:

```
a = 1; b = 2;
```

```
print(a) print (a +
```

```
b)
```

```
print ('сумма = ', a + b)
```

Для команды **print** может задаваться так называемый сепаратор — разделитель между элементами вывода:

x=2

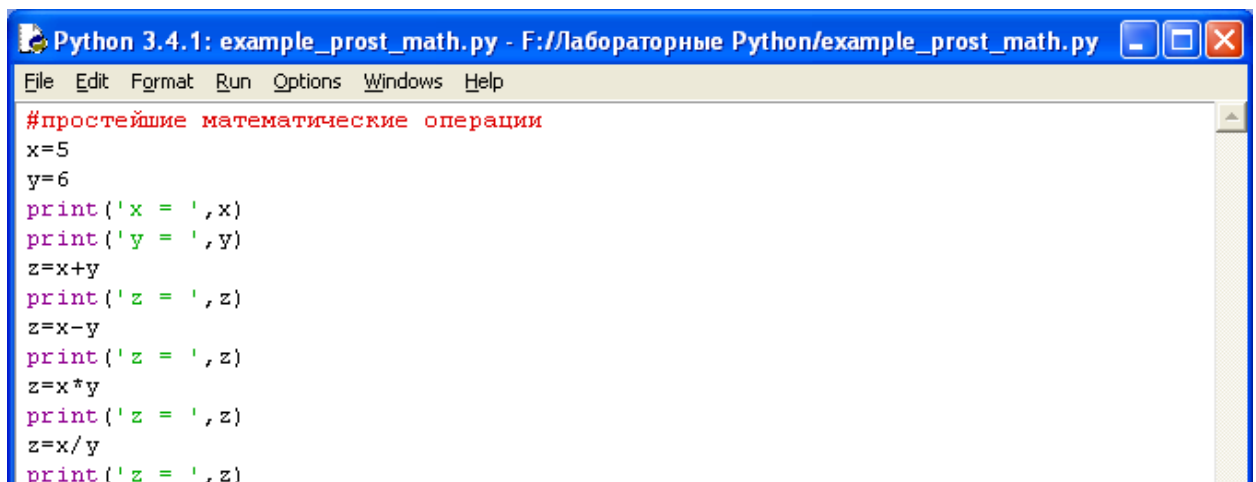
y=5

```
print ( x, "+", y, "=", x+y, sep = " " )
```

Результат отобразится с пробелами между элементами: 2 + 5 = 7

Простые арифметические операции над числами

x + y	Сложение
x - y	Вычитание
x * y	Умножение
x / y	Деление



```
Python 3.4.1: example_prost_math.py - F://лабораторные Python/example_prost_math.py
File Edit Format Run Options Windows Help
#простейшие математические операции
x=5
y=6
print('x = ', x)
print('y = ', y)
z=x+y
print('z = ', z)
z=x-y
print('z = ', z)
z=x*y
print('z = ', z)
z=x/y
print('z = ', z)
```

Пример программы на Python

```
Python 3.4.1 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 10:38:22) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
x = 5
y = 6
z = 11
z = -1
z = 30
z = 0.8333333333333334
>>> |
```

Результат выполнения программы с применением простых арифметических операций

Для форматированного вывода используется **format**:

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

Синтаксис команды **format**:

поле замены := `"{ [имя поля] [!" преобразование] [:" спецификация] }`"

имя поля := `arg_name ("." имя атрибута | "[" индекс "]")*`

преобразование := `"r"` (внутреннее представление) | `"s"` (человеческое представление)

спецификация := см. ниже

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

Идентификаторы могут быть либо индексами аргументов, либо ключами:

```
example_format.py - K:\Лабораторные Python\example_format.py (3.7.1)
File Edit Format Run Options Window Help
x=11
print(x) #вывод без форматирования
print("{:4}".format(x)) #перед значением переменной x будет присутствовать 2 пробела,
                        #так как число 11 занимает 2 знакоместа
```

```

Python 3.7.1 Shell
File Edit Shell Debug Options Window Help
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:05:16) [MSC v.1915 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: K:\Лабораторные Python\example_format.py =====
 11
  11
>>> |

```

В результате выведется число 11, а перед ним два пробела, так как указано использовать для вывода четыре знакоместа.

Или с несколькими аргументами:

```

example_format1.py - K:\Лабораторные Python\example_format1.py (3.7.1)
File Edit Format Run Options Window Help
x=2
print ("{:4d}{:4d}{:4d}".format (x, x, x))

```

```

Python 3.7.1 Shell
File Edit Shell Debug Options Window Help
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:05:16) [MSC v.1915 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: K:\Лабораторные Python\example_format1.py =====
  2  2  2
>>>

```

В итоге каждое из значений выводится из расчета 4

знакоместа. Спецификация формата:

спецификация	:= [[fill]align][sign][#][0][width][,][.precision][type]
заполнитель	:= символ кроме '{' или '}'
выравнивание	:= "<" ">" "=" "^"
знак	:= "+" "-" " "
ширина	:= integer

ТОЧНОСТЬ	:= integer
ТИП	:= "b" "c" "d" "e" "E" "f" "F" "g" "G" "n" "o" "s" "x" "X" "%"

Тип	Значение
'd', 'i', 'u'	Десятичное число.
'o'	Число в восьмеричной системе счисления.
'x'	Число в шестнадцатеричной системе счисления (буквы в нижнем регистре).
'X'	Число в шестнадцатеричной системе счисления (буквы в верхнем регистре).
'e'	Число с плавающей точкой с экспонентой (экспонента в нижнем регистре).
'E'	Число с плавающей точкой с экспонентой (экспонента в верхнем регистре).
'f', 'F'	Число с плавающей точкой (обычный формат).
'g'	Число с плавающей точкой. с экспонентой (экспонента в нижнем регистре), если она меньше, чем -4 или точности, иначе обычный формат.
'G'	Число с плавающей точкой. с экспонентой (экспонента в верхнем регистре), если она меньше, чем -4 или точности, иначе обычный формат.
'c'	Символ (строка из одного символа или число - код символа).
's'	Строка.
'%'	Число умножается на 100, отображается число с плавающей точкой, а за ним знак %.

Для форматирования вещественных чисел с плавающей точкой используется следующая команда:

```
print('{0:.2f}'.format(вещественное число))
```



```
format_chisla.py - К:/Лабораторные Python/format_chisla.py (3.7.1)
File Edit Format Run Options Window Help
x=10
y=7
print("{0:.2f}".format(x/y))
```

В результате выведется число с двумя знаками после запятой.

```
Python 3.7.1 Shell
File Edit Shell Debug Options Window Help
Python 3.7.1 (v3.7.1:260ec2c36a, Oct
1)] on win32
Type "help", "copyright", "credits"
>>>
===== RESTART: К:/Лаборатор
1.43
>>> |
```

Пример

Напишите программу, которая запрашивала бы у пользователя:

1. ФИО ("Ваши фамилия, имя, отчество?")
2. возраст ("Сколько Вам лет?")
3. место жительства ("Где вы живете?") После этого выводила бы три строки: "Ваше имя"

"Ваш

возраст" "Вы

живете в"

Решение

```
a=input('Введите ваши фамилию, имя, отчество ')
b=input('Сколько вам лет? ')
c=input('Где вы живёте? ')
print('Ваше имя ',a)
print('Ваш возраст ',b)
print('Вы живете в ',c)
```

```
Введите ваши фамилию, имя, отчество Иванов Иван Иванович
Сколько вам лет? 15
Где вы живёте? Уссурийск
Ваше имя Иванов Иван Иванович
Ваш возраст 15
Вы живете в Уссурийск
```

Задания для самостоятельной работы

1. Установите Python <https://www.python.org/ftp/python/3.8.5/python-3.8.5.exe>

2. Напишите программу, которая запрашивала бы у пользователя:

Имя, Фамилия, Возраст, Место жительства

3. фамилия, имя ("Ваши фамилия, имя?")

4. возраст ("Сколько Вам лет?")

5. место жительства ("Где вы

живете?") После этого

выводила бы три строки: "Ваши

фамилия, имя"

"Ваш

возраст" "Вы

живете в"

Практическое занятие 2 Математические операции в Python

Цель работы: познакомиться с основными математическими операциями в Python

Язык Python, благодаря наличию огромного количества библиотек для решения разного рода вычислительных задач, сегодня является конкурентом таким пакетам как Matlab и Octave. Запущенный в интерактивном режиме, он, фактически, превращается в мощный калькулятор. В этом уроке речь пойдет об арифметических операциях, доступных в данном языке. Арифметические операции изучим применительно к числам.

Если в качестве операндов некоторого арифметического выражения используются только целые числа, то результат тоже будет целое число. Исключением является операция деления, результатом которой является вещественное число. При совместном использовании целочисленных и вещественных переменных, результат будет вещественным.

В этом уроке речь пойдет об арифметических операциях, доступных в данном языке.

Если в качестве операндов некоторого арифметического выражения используются только целые числа, то результат тоже будет целое число. Исключением является операция деления, результатом которой является вещественное число. При совместном использовании целочисленных и вещественных переменных, результат будет вещественным.

Целые числа (int)

Числа в Python 3 поддерживают набор самых обычных математических операций:

$x + y$	Сложение
$x - y$	Вычитание
$x * y$	Умножение
x / y	Деление

<code>x // y</code>	Получение целой части от деления
<code>x % y</code>	Остаток от деления
<code>-x</code>	Смена знака числа
<code>abs(x)</code>	Модуль числа
<code>divmod(x, y)</code>	Пара (<code>x // y</code> , <code>x % y</code>)
<code>x ** y</code>	Возведение в степень
<code>pow(x, y[, z])</code>	<p><code>x</code> : Число, которое требуется возвести в степень. <code>y</code> : Число, являющееся степенью, в которую нужно возвести первый аргумент. Если число отрицательное или одно из чисел "<code>x</code>" или "<code>y</code>" не целые, то аргумент "<code>z</code>" не принимается. <code>z</code> : Число, на которое требуется произвести деление по модулю. Если число указано, ожидается, что "<code>x</code>" и "<code>y</code>" положительны и имеют тип <code>int</code>.</p>

Пример применения вышеописанных операций над целыми числами

```

x = 5
y = 2
z = 3
x+y = 7
x-y = 3
x*y = 10
x/y = 2.5
x//y = 2
x%y = 1
-x = -5
abs(-x) = 5
divmod(x, y) = (2, 1)
x**y = 25
pow(x, y, z) = 1

```

Вещественные числа (float)

Вещественные числа поддерживают те же операции, что и целые. Однако (из-за представления чисел в компьютере) вещественные числа неточны, и это может привести к ошибкам.

Пример применения вышеописанных операций над вещественными числами

```

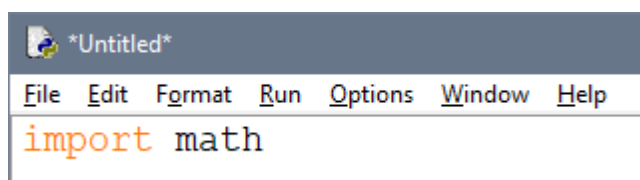
x = 5.5
y = 2.3
x+y = 7.8
x-y = 3.2
x*y = 12.649999999999999
x/y = 2.3913043478260874
x//y = 2.0
x%y = 0.9000000000000004
-x = -5.5
abs(-x) = 5.5
divmod(x,y) = (2.0, 0.9000000000000004)
x**y = 50.44686540422945

```

Библиотека (модуль) math

В стандартную поставку Python входит библиотека math, в которой содержится большое количество часто используемых математических функций.

Для работы с данным модулем его предварительно нужно импортировать.



Рассмотрим наиболее часто используемые функции модуля math

math.ceil(x)	Возвращает ближайшее целое число большее, чем x
math.fabs(x)	Возвращает абсолютное значение числа x
math.factorial(x)	Вычисляет факториал x
math.floor(x)	Возвращает ближайшее целое число меньшее, чем x
math.exp(x)	Вычисляет e**x
math.log2(x)	Логарифм по основанию 2
math.log10(x)	Логарифм по основанию 10
math.log(x[, base])	По умолчанию вычисляет логарифм по основанию e, дополнительно можно указать основание логарифма
math.pow(x, y)	Вычисляет значение x в степени y

<code>math.sqrt(x)</code>	Корень квадратный от x
---------------------------	------------------------

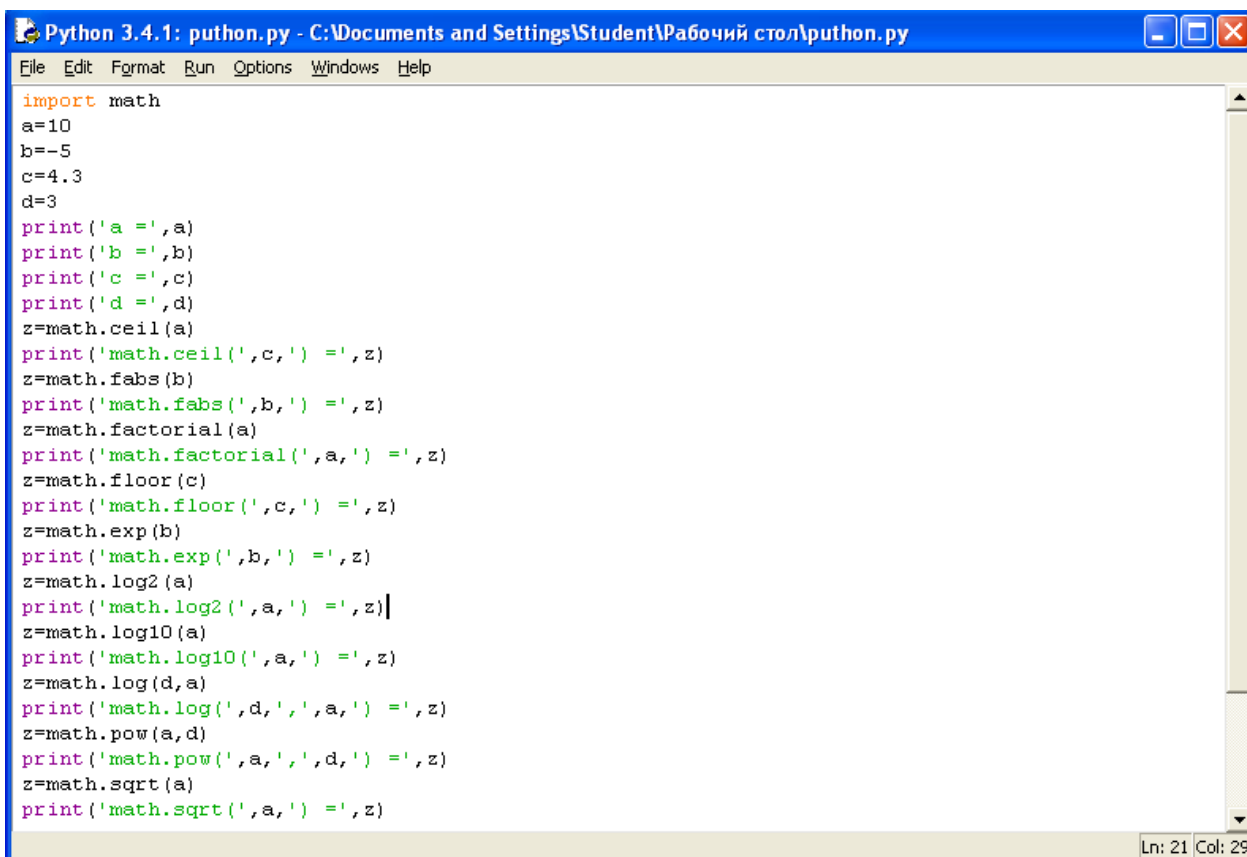
Пример применения вышеописанных функций над числами

В программе определены 4 переменные - a, b, c, d, каждая из которых является либо целым числом, либо вещественным, либо отрицательным.

Командой `print()` выводится значение каждой переменной на экран при выполнении программы.

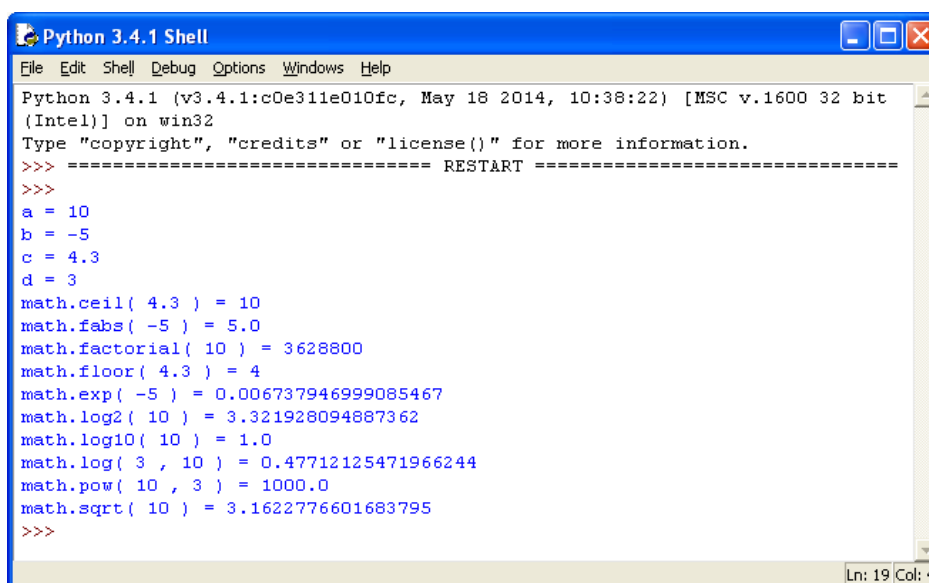
В переменную `z` помещается результат выполнения функции модуля `math`.

Затем командой `print()` выводится сообщение в виде используемой функции и её аргумента и результат её выполнения.



```
Python 3.4.1: puthon.py - C:\Documents and Settings\Student\Рабочий стол\puthon.py
File Edit Format Run Options Windows Help
import math
a=10
b=-5
c=4.3
d=3
print('a =', a)
print('b =', b)
print('c =', c)
print('d =', d)
z=math.ceil(a)
print('math.ceil(' ,c,') =', z)
z=math.fabs(b)
print('math.fabs(' ,b,') =', z)
z=math.factorial(a)
print('math.factorial(' ,a,') =', z)
z=math.floor(c)
print('math.floor(' ,c,') =', z)
z=math.exp(b)
print('math.exp(' ,b,') =', z)
z=math.log2(a)
print('math.log2(' ,a,') =', z)
z=math.log10(a)
print('math.log10(' ,a,') =', z)
z=math.log(d, a)
print('math.log(' ,d,',' ,a,') =', z)
z=math.pow(a, d)
print('math.pow(' ,a,',' ,d,') =', z)
z=math.sqrt(a)
print('math.sqrt(' ,a,') =', z
Ln: 21 Col: 29
```

Пример программы на Python



```
Python 3.4.1 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.1 [v3.4.1:c0e311e010fc, May 18 2014, 10:38:22] [MSC v.1600 32 bit
(Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
a = 10
b = -5
c = 4.3
d = 3
math.ceil( 4.3 ) = 10
math.fabs( -5 ) = 5.0
math.factorial( 10 ) = 3628800
math.floor( 4.3 ) = 4
math.exp( -5 ) = 0.006737946999085467
math.log2( 10 ) = 3.321928094887362
math.log10( 10 ) = 1.0
math.log( 3 , 10 ) = 0.47712125471966244
math.pow( 10 , 3 ) = 1000.0
math.sqrt( 10 ) = 3.1622776601683795
>>>
Ln: 19 Col: 4
```

Результат выполнения программы с применением функций модуля math

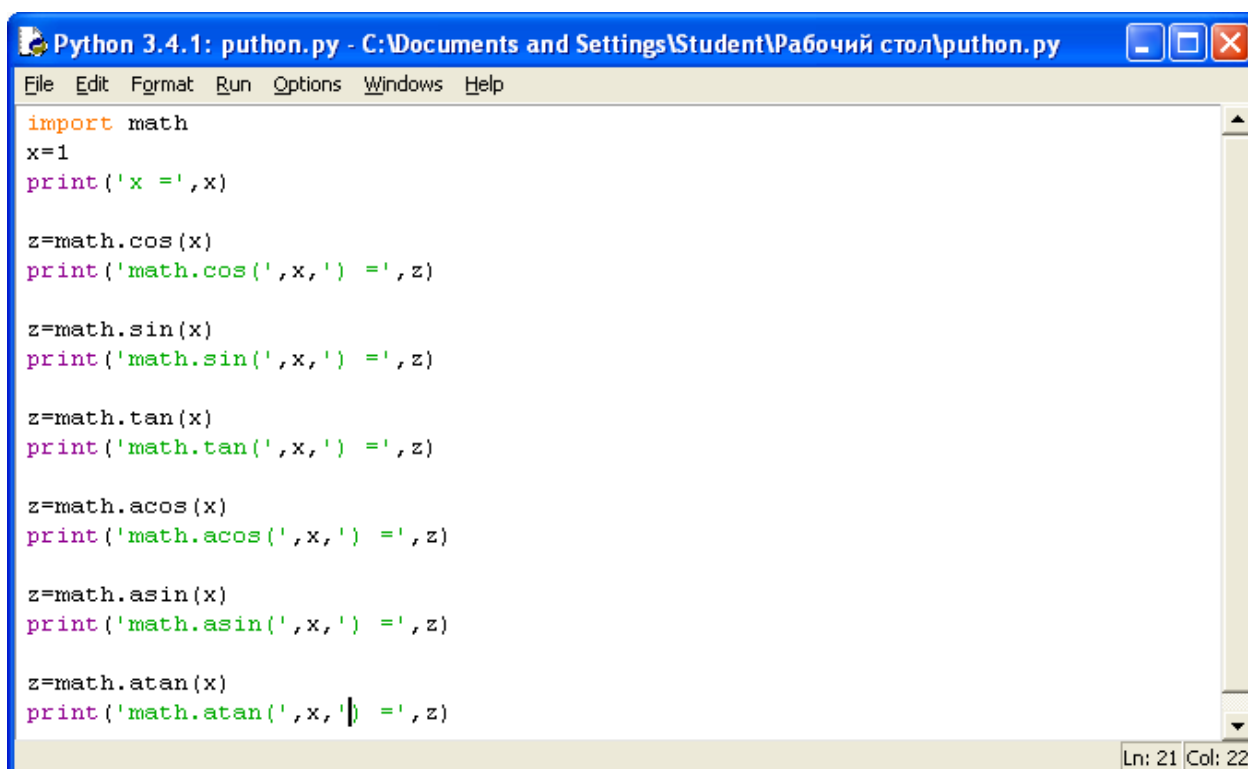
Тригонометрические функции модуля math

math.cos(x)	Возвращает cos числа X
math.sin(x)	Возвращает sin числа X
math.tan(x)	Возвращает tan числа X
math.acos(x)	Возвращает acos числа X
math.asin(x)	Возвращает asin числа X
math.atan(x)	Возвращает atan числа X

Пример применения вышеописанных функций над числами В программе определена переменная x, содержащая целое число. Значение переменной выводится командой print() на экран.

В переменную z помещается результат выполнения тригонометрической функции модуля math.

Затем командой print() выводится сообщение в виде используемой функции и её аргумента и результат её выполнения.



```
Python 3.4.1: puthon.py - C:\Documents and Settings\Student\Рабочий стол\puthon.py
File Edit Format Run Options Windows Help
import math
x=1
print('x =', x)

z=math.cos(x)
print('math.cos(', x, ') =', z)

z=math.sin(x)
print('math.sin(', x, ') =', z)

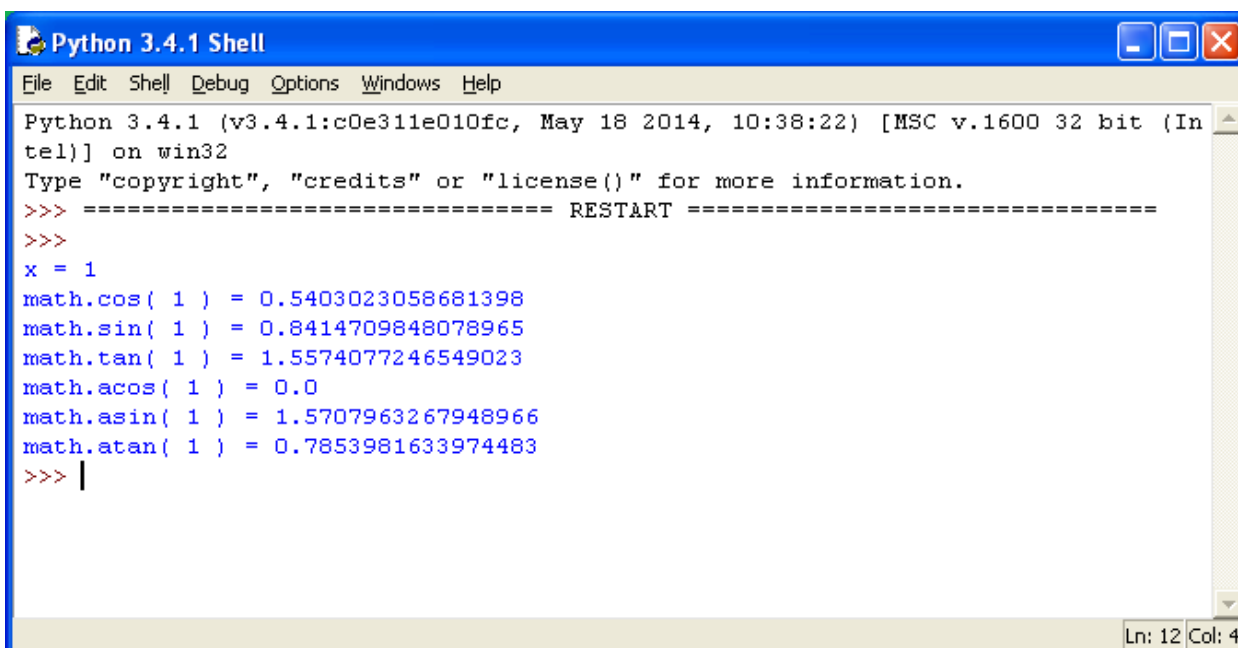
z=math.tan(x)
print('math.tan(', x, ') =', z)

z=math.acos(x)
print('math.acos(', x, ') =', z)

z=math.asin(x)
print('math.asin(', x, ') =', z)

z=math.atan(x)
print('math.atan(', x, ') =', z)
Ln: 21 Col: 22
```

Пример программы с использованием тригонометрических функций модуля `math`



```
Python 3.4.1 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 10:38:22) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
x = 1
math.cos( 1 ) = 0.5403023058681398
math.sin( 1 ) = 0.8414709848078965
math.tan( 1 ) = 1.5574077246549023
math.acos( 1 ) = 0.0
math.asin( 1 ) = 1.5707963267948966
math.atan( 1 ) = 0.7853981633974483
>>> |
```

Результат выполнения программы с применением тригонометрических функций модуля `math`

Константы:

1. **`math.pi`** - число π .
2. **`math.e`** - число e (экспонента).

Пример

Напишите программу, которая бы вычисляла заданное арифметическое выражение при заданных переменных. Ввод переменных осуществляется с клавиатуры. Вывести результат с 2-мя знаками после запятой.

Задание

$$Z = \frac{9\pi t + 10 \cos(x)}{\sqrt{t} - |\sin(t)|} * e^x$$

```
x=10;
t=1
```

Решение

Сначала импортируем модуль `math`. Для этого воспользуемся командой `import math`.

Затем следует ввести значения двух переменных целого типа `x` и `t`. Для ввода данных используется команда `input`, но так как в условии даны целые числа, то нужно сначала определить тип переменных: `x=int()`, `t=int()`.

Определив тип переменных, следует их ввести, для этого в скобках команды `int()` нужно написать команду `input()`.

Для переменной `x` это выглядит так: `x=int(input("сообщение при вводе значения"))`.

Для переменной `t` аналогично: `t=int(input("сообщение при вводе значения"))`. Следующий шаг - это составление арифметического выражения, результат которого поместим в переменную `z`.

Сначала составим числитель. Выглядеть он будет так: `9*math.pi*t+10*math.cos(x)`.

Затем нужно составить знаменатель, при этом обратим внимание на то, что числитель делится на знаменатель, поэтому и числитель и знаменатель нужно поместить в скобки `()`, а между ними написать знак деления `/`.

Выглядеть это будет так:
`(9*math.pi*t+10*math.cos(x))/(math.sqrt(t)- math.fabs(math.sin(t)))`.

Последним шагом является умножение дроби на экспоненту в степени `x`.

Так как умножается вся дробь, то следует составленное выражение поместить в скобки `()`, а уже потом написать функцию `math.pow(math.e,x)`.

В результате выражение будет иметь вид:

```
z=((9*math.pi*t+10*math.cos(x))/(math.sqrt(t)-
)-
math.fabs(math.sin(t))))*math.pow(math.e,x).
```

При составлении данного выражения следует обратить внимание на количество открывающихся и закрывающихся скобок.

Командой `print()` выведем значение переменной, отформатировав его командой `format`.

Сам формат записывается в апострофах в фигурных скобках `{}`.

В задаче требуется вывести число с двумя знаками после запятой, значит вид формата будет выглядеть следующим образом: `{0:.2f}`, где 2 - это количество знаков после запятой, а f указывает на то, что форматируется вещественное число. При этом перед 2 нужно поставить точку, указав тем самым на то, что форматируем именно дробную часть числа.

```
example_math_var0.py - C:/Users/maxim/Desktop/Python/example_math_var0.py (3.7.1)
File Edit Format Run Options Window Help
import math
x=int(input("Введите переменную x:"))
t=int(input("Введите переменную t:"))
z=((9*math.pi*t+10*math.cos(x))/(math.sqrt(t)-math.fabs(math.sin(t))))*math.pow(math.e,x)
print("z = {0:.2f}".format(z))
```

Результат

```
Python 3.7.1 Shell
File Edit Shell Debug Options Window Help
Python 3.7.1 (v3.7.1:260ec2c36a,
1) on win32
Type "help", "copyright", "credit
>>>
===== RESTART: C:/Users/maxim/
Введите переменную x:10
Введите переменную t:1
z = 2762685.71
>>> |
```

Задания для самостоятельной работы

Воспроизвести задание из примера. Сделать скриншоты кода и результата.

Практическое занятие 3 Структура ветвление в Python

Цель работы: познакомиться со структурой ветвление (if, if-else, if-elif-else).

Научиться работать с числами и строками используя данную структуру.

Условный оператор ветвления if, if-else, if-elif-else

Оператор ветвления if позволяет выполнить определенный набор инструкций в зависимости от некоторого условия. Возможны следующие варианты использования.

1. Конструкция if

Синтаксис оператора if выглядит так:

if логическое выражение:

```
команда_1  
команда_2  
...  
команда_n
```

После оператора if записывается логическое выражение.

Логическое выражение — конструкция языка программирования, результатом вычисления которой является «истина» или «ложь».

Если это выражение истинно, то выполняются инструкции, определяемые данным оператором. Выражение является истинным, если его результатом является число не равное нулю, непустой объект, либо логическое True. После выражения нужно поставить двоеточие “:”.

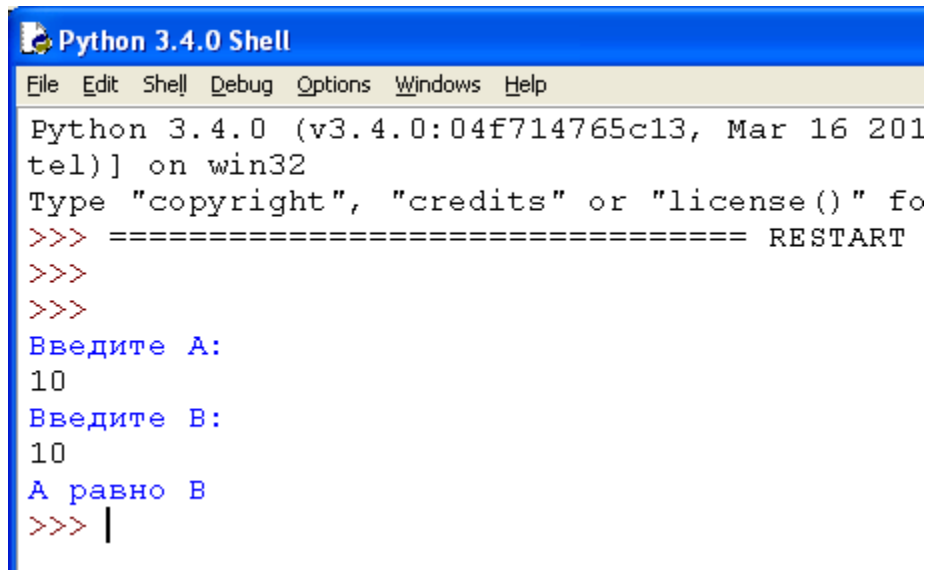
ВАЖНО: блок кода, который необходимо выполнить, в случае истинности выражения, отделяется **четырьмя** пробелами слева!

Программа запрашивает у пользователя два числа, затем сравнивает их и если числа равны, то есть логическое выражение $A==B$ истинно, то выводится

соответствующее сообщение.

```
Python 3.4.0: example_if.py - F:/example_if.py
File Edit Format Run Options Windows Help
print ('Введите A:')
A=input ()
print ('Введите B:')
B=input ()
if A==B:
    print ('A равно B')
```


Пример программы на Python



```
Python 3.4.0 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.0 (v3.4.0:04f714765c13, Mar 16 201
tel)] on win32
Type "copyright", "credits" or "license()" fo
>>> ===== RESTART
>>>
>>>
Введите A:
10
Введите B:
10
A равно B
>>> |
```

Результат выполнения программы с использованием условного оператора if

2. Конструкция if - else

Бывают случаи, когда необходимо предусмотреть альтернативный вариант выполнения программы. Т.е. при истинном условии нужно выполнить один набор инструкций, при ложном – другой. Для этого используется конструкция if – else.

Синтаксис оператора if – else выглядит так:

if логическое выражение:

команда_1

команда_2

...

команда_n

else:

команда_1

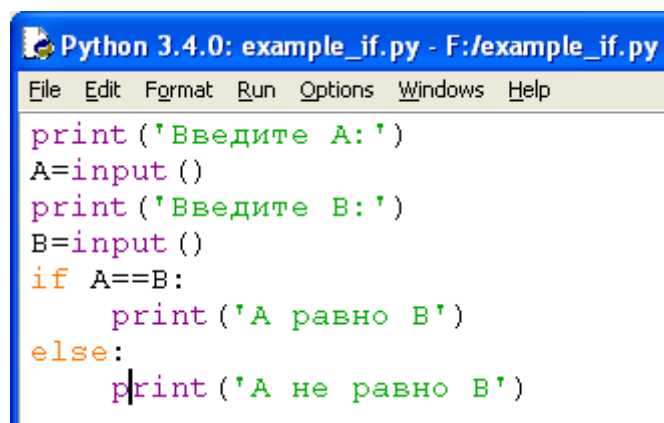
команда_2

...

команда_n

Программа запрашивает у пользователя два числа, затем сравнивает их и если числа равны, то есть логическое

выражение $A==B$ истинно, то выводится соответствующее сообщение. В противном случае выводится сообщение, что числа не равны.



```
Python 3.4.0: example_if.py - F:/example_if.py
File Edit Format Run Options Windows Help
print ('Введите A:')
A=input ()
print ('Введите B:')
B=input ()
if A==B:
    print ('A равно B')
else:
    print ('A не равно B')
```

Пример программы на Python

```
>>> ===== RESTART
>>>
Введите A:
10
Введите B:
5
A не равно B
>>> |
```

Результат выполнения программы с использованием условного оператора if-else

3. Конструкция if - elif - else

Для реализации выбора из нескольких альтернатив можно использовать конструкцию if – elif – else.

Синтаксис оператора if – elif – else выглядит так:

```
if логическое выражение_1:
    команда_1
elif логическое выражение_2:
    команда_2
...
else:
    команда_n
```

```

    команда_n
elif логическое выражение_2:
    команда_1
    команда_2
    ...

    команда_n
elif логическое выражение_3:

    команда_1
    команда_2
    ...

    команда_n

else:
    команда_1
    команда_2
    ...

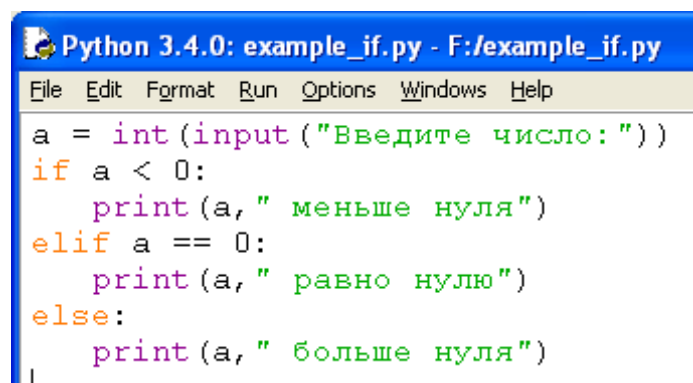
    команда_n

```

Программа запрашивает число у пользователя и сравнивает его с нулём $a < 0$.

Если оно меньше нуля, то выводится сообщение об этом. Если первое логическое выражение не истинно, то программа переходит ко второму - $a == 0$. Если оно истинно, то программа выведет сообщение, что число равно нулю, в противном случае, если оба вышеуказанных логических выражения оказались ложными, то программа выведет сообщение, что введённое число больше

нуля.



```

Python 3.4.0: example_if.py - F:/example_if.py
File Edit Format Run Options Windows Help
a = int(input("Введите число:"))
if a < 0:
    print(a, " меньше нуля")
elif a == 0:
    print(a, " равно нулю")
else:
    print(a, " больше нуля")
|

```

Пример программы на Python

```

Введите число: 41
41 больше нуля
>>> ===== RESTART =
>>>
Введите число: -5
-5 меньше нуля
>>> ===== RESTART =
>>>
Введите число: 0
0 равно нулю
>>>

```

Результат выполнения программы с использованием условного оператора if-elif-else

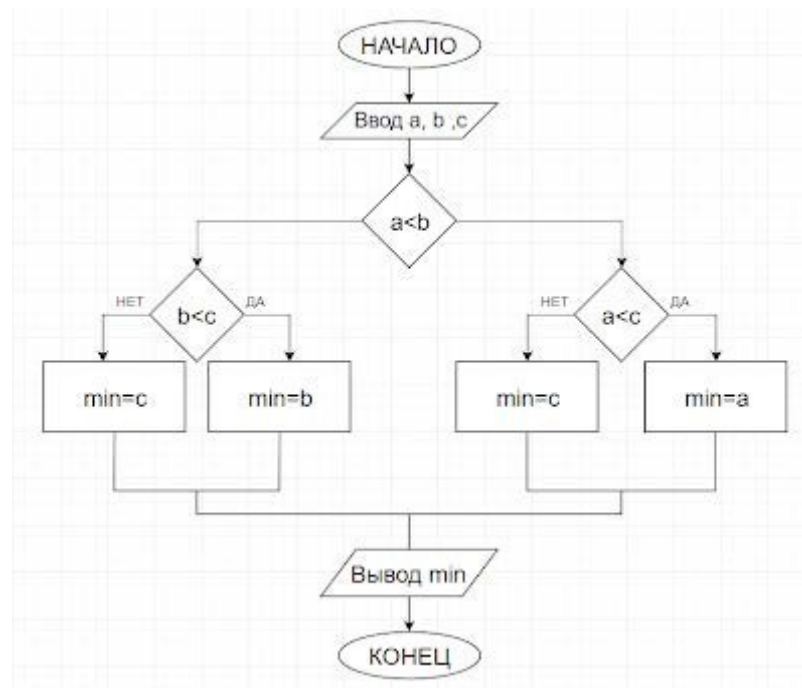
При
мер

Задание

Дано 3 числа. Найти минимальное среди них и вывести на экран.

Решение

Для простоты построим блок-схему задачи.



Командами

```
a=input()  
b=input()  
c=input()
```

введём три числа, присвоив значения переменным a, b, c.

Условной конструкцией if-else проверим на истинность логическое выражение $a < b$. Если оно истинно, то переходим на проверку логического выражения $a < c$. Если оно истинно, то переменной "y" присвоим значение переменной "a", т.е. "a" будет минимальным, а иначе "y" присвоится значение переменной "c".

Если в начале логическое выражение $a < b$ оказалось ложным, то переходим на проверку другого логического выражения $b < c$.

Если оно истинно, то "y" присвоится значение переменной "b", иначе "c".

Командой print() выводим минимальное значение.

```
#нахождение минимального из 3-х чисел  
a=input('Введите целое число \n')  
b=input('Введите целое число \n')  
c=input('Введите целое число \n')  
if a<b:  
    if a<c:  
        y=a  
    else:  
        y=c  
else:  
    if b<c:  
        y=b  
    else:  
        y=c  
print('Минимальное:', y)
```

Пример программы

```
Введите целое число  
2  
Введите целое число  
5  
Введите целое число  
1  
Минимальное: 1
```

Результат выполнения программы

Задания для самостоятельной работы

Задание

Даны три целых числа. Выбрать из них те, которые принадлежат интервалу $[1,3]$.

Практическое занятие 4 Работа с циклами в Python

Цель работы: познакомиться с циклическими конструкциями В Python существуют два типа циклических выражений:

1. Цикл while
2. Цикл for

1. Цикл while в Python

Инструкция while в Python повторяет указанный блок кода до тех пор, пока указанное в цикле логическое выражение будет оставаться истинным.

Синтаксис цикла while:

while логическое выражение:

```
команда 1  
команда 2  
...  
команда n
```

После ключевого слова while указывается условное выражение, и пока это выражение возвращает значение True, будет выполняться блок инструкций, который идет далее.

Все инструкции, которые относятся к циклу while, располагаются на последующих строках и должны иметь отступ от начала строки

(4
пробела).

```
#!/ Программу по вычислению факториала  
number = int(input("Введите число: "))  
i = 1  
factorial = 1  
while i <= number:  
    factorial *= i  
    i += 1  
print("Факториал числа", number, "равен", factorial)
```

Пример программы на Python

```

Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 10:38:22) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
Введите число: 5
Факториал числа 5 равен 120
>>> |

```

Результат выполнения программы с использованием циклического оператора `while`

2. Цикл `for` в Python:

Цикл `for` в Python обладает способностью перебирать элементы любого комплексного типа данных (например, строки или списка).

Синтаксис цикла `for`:

for `int in range()`:

команда 1

команда 2

...

команда n

Переменной `int` присваивается значение первого элемента функции `range()`, после чего выполняются команды. Затем переменной `int` присваивается следующее по порядку значение и так далее до тех пор, пока не будут перебраны все элементы функции `range()`.

Функция `range()` является универсальной функцией Python для создания списков (list) содержащих арифметическую прогрессию.

Чаще всего она используется в циклах `for`.

`range(старт, стоп, шаг)` - так выглядит стандартный вызов функции `range()` в Python. По умолчанию старт равняется нулю, шаг единице.

Пример.

1. Найти сумму `n` элементов следующего ряда чисел: 1 -0.5 0.25 - 0.125 ...
- n. Количество элементов (`n`) вводится с клавиатуры. Вывести на экран каждый член ряда и его сумму. Решить задачу используя циклическую конструкцию `for`.

Решение:

В данном случае ряд чисел состоит из элементов, где каждый следующий меньше предыдущего в два раза по модулю и имеет обратный знак. Значит, чтобы получить следующий элемент, надо предыдущий разделить на -2.

Какой-либо переменной надо присвоить значение первого элемента ряда (в данном случае это 1). Далее в цикле добавлять ее значение к переменной, в которой накапливается сумма, после чего присваивать ей значение следующего элемента ряда, разделив текущее значение на -2. Цикл должен выполняться n раз.

```
n=int(input('Введите количество элементов последовательности: '))
x=1
s=0
print(x)
for i in range(n):
    s+=x
    x/=-2
    print(x)
print('Сумма ряда:',s)
```

Пример программы с циклом for

```
Введите количество элементов последовательности: 5
1
-0.5
0.25
-0.125
0.0625
-0.03125
Сумма ряда: 0.6875
```

Результат выполнения программы

2. Дано целое число, не меньшее 2. Выведите его наименьший натуральный делитель, отличный от 1.

Решение:

Для начала введём целое число командой `int(input(текст сообщения))`. Затем зададим переменной `i` значение 2. Переменная `i` выполняет роль счётчика. Если задать ей значение 1, то условие задачи не будет

выполнено, а результатом всегда будет 1.

В цикле `while` в качестве логического выражения используется команда `n%i` сравниваемая с нулём. Таким образом, если остаток от деления введённого числа на текущее значение `i` не равно нулю, то счётчик увеличивается на 1, а если равно нулю цикл заканчивается и командой `print()` выводится сообщение и значение `i`.

```
n = int(input('Введите целое число не меньше 2\n'))
i = 2
while n%i != 0:
    i+=1
print('наименьший натуральный делитель:', i)
```

Пример программы с циклом while

```
Введите целое число не меньше 2
49
наименьший натуральный делитель: 7
```

Результат выполнения программы

Задание.

1. Дано вещественное число – цена 1 кг конфет. Вывести стоимость 1, 2, ... 10 кг конфет. Решить задачу используя циклическую конструкцию for.
2. Дана непустая последовательность целых чисел, оканчивающаяся нулем. Найти: а) сумму всех чисел последовательности; б) количество всех чисел последовательности
Решить задачу используя циклическую конструкцию while.

Практическое занятие 5 Работа со строками в Python

Цель работы: познакомится с методами работы со строками.

Учащийся должен:

Владеть:

Навыками составления линейных алгоритмов на языке программирования Python с использованием строковых данных;

Уметь:

Применять функции и методы строк при обработке строковых данных;

Знать:

Операции и методы обработки строк.

Строка — базовый тип представляющий из себя неизменяемую последовательность символов; str от «string» — «строка».

Функции и методы работы со строками

Функция или метод	Назначение
S1 + S2	Конкатенация (сложение строк)
S1 * 3	Повторение строки
S[i]	Обращение по индексу
S[i:j:step]	Извлечение среза
len(S)	Длина строки
S.join(список)	Соединение строк из последовательности str через разделитель, заданный строкой
S1.count(S[, i, j])	количество вхождений подстроки s в строку s1. Результатом является число. Можно указать позицию начала поиска i и окончания поиска j
S.find(str, [start], [end])	Поиск подстроки в строке. Возвращает номер первого вхождения или -1
S.index(str, [start], [end])	Поиск подстроки в строке. Возвращает номер первого вхождения или вызывает ValueError

[end])	
S.rind ex (str, [start], [end])	Поиск подстроки в строке. Возвращает номер последнего вхождения или вызывает ValueError
S.replace (шаблон, замена)	Замена шаблона
S.split (символ)	Разбиение строки по разделителю
S.upper ()	Преобразование строки к верхнему регистру
S.lower ()	Преобразование строки к нижнему регистру

Ниже приведена программа, демонстрирующая использование функций и методов работы со строками.

```

example_string.py - K:\Лабораторные Python\example_string.py (3.7.1)
File Edit Format Run Options Window Help
s1="Пропаганда"
s2="Сенсация"
s3="Сенсация*Сенсация*Сенсация*Сенсация"
s4='OxOxOxOx'
print('s1 = ',s1)
print('s2 = ',s2)
print('s3 = ',s3)
print('s4 = ',s4)
print('s1+s2 = ',s1+s2) #сложение двух строк
print('s1*3 = ',s1*3) #умножение строки на 3, т.е.строка выведется 3 раза
print('s1[2] = ',s1[2]) #вывод элемента строки s1 с индексом 2
print('s1[2:4] = ',s1[2:4]) #извлечение среза строки s1 начиная с индекса 2
#и заканчивая индексом 4
print('s3.count = ',s3.count(s2)) #количество вхождений подстроки s2 в S3,
#в результате выведется число
print('s1.find('a') = ',s1.find('a')) #поиск подстроки 'a' в строке s1
#результатом будет номер первого вхождения
print('s1.index('n') = ',s1.index('n'))#поиск подстроки 'n' в строке s1
#результатом будет номер первого вхождения
print('s1.rindex('d') = ',s1.rindex('a'))#поиск подстроки 'a' в строке s1
#возвращает номер последнего вхождения
print('s4.replace('Ox','Ax',2) = ',s4.replace('Ox','Ax',2))#замена шаблона. Строка 'Ox' - это шаблон
#строка 'Ax' - это замена
#в строке 4 последовательность 'Ox' будет заменена
#на 'Ax' с шагом 2
print('s3.split('*') = ',s3.split('*'))#разбиение по разделителю *
print('s1.upper = ',s1.upper())#перевод символов в верхний регистр
print('s1.lower = ',s1.lower())#перевод символов в нижний регистр
Ln: 20 Col: 40

```

Пример программы на Python

```
Python 3.7.1 Shell
File Edit Shell Debug Options Window Help
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:05:16) [MSC v.191] on win32
Type "help", "copyright", "credits" or "license()" for more
>>>
===== RESTART: К:\Лабораторные Python\example_string.py =====
s1 = Пропаганда
s2 = Сенсация
s3 = Сенсация*Сенсация*Сенсация*Сенсация
s4 = ОхОхОхАх
s1+s2 = ПропагандаСенсация
s1*3 = ПропагандаПропагандаПропаганда
s1[2] = о
s1[2,4] = оп
s3.count = 4
s1.find(a) = 4
s1.index(п) = 3
s1.rindex(д) = 9
s4.replace(Ох,Ах,2) = АхАхОхАх
s3.split(*) = ['Сенсация', 'Сенсация', 'Сенсация', 'Сенсация']
s1.upper = ПРОПАГАНДА
s1.lower = пропаганда
```

Результат выполнения программы с использованием функций и методов работы со строками

Пример

Вариант

0

Проверить, будет ли строка читаться одинаково справа налево и слева направо (т. е. является ли она палиндромом).

Решение

Сначала введём строку командой: `s=input('Введите строку ')`.

Затем определим логическую переменную `flag` и присвоим ей значение 1: `flag=1`.

Для начала в введённой строке нужно удалить пробелы. Для этого воспользуемся циклической конструкцией `for`, которая выполнится столько раз, какую имеет длину строка. Длину строки определим функцией `len(s)`.

В теле цикла будем проверять следующее условие: `s[i]!=' '`. Данное логическое выражение будет истинно в том случае, если `i`-ый элемент строки не будет равен пробелу, тогда выполнится команда следующая после двоеточия: `string+=s[i]`.

К строке `string`, которая была объявлена в начале программы, будет добавляться посимвольно строка `s`, но уже без пробелов.

Для проверки строки на "палиндром" воспользуемся циклической конструкцией `for`.

Длина половины строки находится делением нацело на 2. Если количество символов нечетно, то стоящий в середине не учитывается, т.к. его сравниваемая пара - он сам.

Количество повторов цикла равно длине половины строки. Длину строки определим функцией `len(s)`, где аргумент введенная нами строка `s`. Зная длину строки, можно вычислить количество повторов цикла. Для этого целочисленно разделим длину строки на 2: `len(s)//2`.

Для задания диапазона для цикла используем функцию `range()`, в которой аргументом будет являться половина длины строки: `range(len(s)//2)`.

```
for i in range(len(s)//2):
```

Если символ с индексом `i` не равен "симметричному" символу с конца строки (который находится путем индексации с конца)

```
if s[i] != s[-1-i],
```

то переменной `flag` присваивается значение 0 и происходит выход из цикла командой `break`.

Далее, при помощи условной конструкции `if-else` в зависимости от значения `flag` либо - 0, либо -1 выводится сообщение, что строка палиндром, либо нет.

```
s=input('Введите строку \n')
flag=1
string=''
for i in range(len(s)):
    if s[i]!=' ':
        string+=s[i]
print(string)
for i in range(len(s)//2):
    if string[i]!=string[-i-1]:
        flag=0
        break
if flag: print('Палиндром')
else: print('не палиндром')
```

Пример программы на Python

```
Введите строку
а роза упала на лапу азора
арозаупаланалапуазора
Палиндром
```

Результат выполнения программы

Задания для самостоятельной работы (по вариантам)

Вариант 1

Дана строка, содержащая русскоязычный текст. Найти количество слов, начинающихся с буквы "е".

Вариант 2

В строке заменить все двоеточия (:) знаком процента (%).
Подсчитать количество замен.

Вариант 3

В строке удалить символ точку (.) и подсчитать количество удаленных символов.

Вариант 4

В строке заменить букву(а) буквой (о). Подсчитать количество замен. Подсчитать, сколько символов в строке.

Вариант 5

В строке заменить все заглавные буквы строчными.

Вариант 6

В строке удалить все буквы "а" и подсчитать количество удаленных символов.

Вариант 7

Дана строка. Преобразовать ее, заменив звездочками все буквы "п", встречающиеся среди первых $n/2$ символов. Здесь n - длина строки.

Вариант 8

Дана строка, заканчивающаяся точкой. Подсчитать, сколько слов в строке.

Вариант 9

Определить, сколько раз в тексте встречается заданное слово.

Вариант 10

Дана строка-предложение на английском языке. Преобразовать строку так, чтобы каждое слово начиналось с заглавной буквы.

Вариант 11

Дана строка. Подсчитать самую длинную последовательность подряд идущих букв «н». Преобразовать ее, заменив точками все восклицательные знаки.

Вариант 12

Дана строка. Вывести все слова, оканчивающиеся на букву "я".

Вариант 13

Дана строка символов, среди которых есть одна открывающаяся и одна закрывающаяся скобки. Вывести на экран все символы, расположенные внутри этих скобок.

Вариант 14

Дана строка. Вывести все слова, начинающиеся на букву "а" и слова оканчивающиеся на букву "я".

Вариант 15

Дана строка текста. Подсчитать количество букв «т» в строке.

Практическое занятие 6 Работа со списками. Операции над списками в Python

Цель работы: Изучение одномерных массивов в Python.

Массивы (списки) в Python — это определенное количество элементов одного типа, которые имеют общее имя, и каждый элемент имеет свой индекс — порядковый номер.

Часто для работы с массивами используются списки.

Список (list) – это структура данных для хранения объектов различных типов.

Списки являются упорядоченными последовательностями, которые состоят из различных типов данных, заключающихся в квадратные скобки [] и отделяющиеся друг от друга с помощью запятой.

Создание списков на Python.

Создать список можно несколькими способами

1. Получение списка через присваивание конкретных значений.

Так выглядит в коде Python пустой список:

```
s = [] # Пустой список
```

Примеры создания списков со значениями:

```
l=[5, 75, -4, 7, -51] # список целых чисел
l=[1.13, 5.34, 12.63, 4.6, 34.0, 12.8] # список из вещественных чисел
l=["Оля", "Владимир", "Михаил", "Дарья"] # список из строк
l=["Москва", "Иванов", 12, 124] # смешанный список
l=[[0, 0, 0], [1, 0, 1], [1, 1, 0]] # список, состоящий из списков
l=['s', 'p', ['isok'], 2] # список из значений и списка
```

Списки можно складывать (конкатенировать) с помощью знака «+»:

```
l=[1, 3]+[4,23]+[5]
print('l=[1, 3]+[4,23]+[5] =', l)
```

Результат:

```
>>>
l=[1, 3]+[4,23]+[5] = [1, 3, 4, 23, 5]
>>> |
```

2. Создание списка при помощи функции `Split()`.

Используя функцию `split` в Python можно получить из строки список. `stroka = "Привет, страна"`

```
lst=stroka.split(",")
```

```
stroka = "Здравствуй, Дедушка Мороз" #stroka - строка
lst=stroka.split(",") #lst - список
print('stroka = ',stroka)
print('lst=stroka.split(","):',lst)
```

Результат:

```
===== RESTART: C:/Users/maxim/Desktop/ex_list_
stroka = Здравствуй, Дедушка Мороз
lst=stroka.split(","): ['Здравствуй', ' Дедушка Мороз']
```

3. Генераторы списков.

В Python создать список можно также при помощи генераторов.

Первый способ.

Сложение одинаковых списков заменяется умножением:

Список из 10 элементов, заполненный

единицами `l = [1]*10`

Второй

способ.

Пример 1.

```
l = [i for i in range(10)]
```

Пример 2.

```
c = [c * 3 for c in 'list']
print(c) # ['lll', 'iii', 'sss', 'ttt']
```

```

Создание списка из строки.
l = list (строка):
 ['c', 'т', 'p', 'o', 'к', 'a']

Создание списка при помощи функции Split().
stroka=" Hello, friend "
lst=stroka.split(","):
 ['Hello', ' friend']

Генераторы списков.
Первый способ.
l = [1]*10:
 [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

Второй способ. Пример 1.
l = [i for i in range(10)]:
 [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Второй способ. Пример 2.
c=[c*3 for c in "list"]:
 ['lll', 'iii', 'sss', 'ttt']

```

Примеры использования генераторов списка. Пример 1.

Заполнить список квадратами чисел от 0 до 9, используя генератор списка. Решение:

```
l = [i*i for i in range(10)]
```

Пример 2.

Заполнить список числами, где каждое последующее число больше на 2.

```
2. l = [(i+1)+i for i in range(10)]
```

```
print(l)
```

```

Заполнить список квадратами чисел от 0 до 9, используя генератор списка.
l = [i*i for i in range(10)]:
 [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

Заполнить список числами, где каждое последующее число больше на 2.
l = [(i+1)+i for i in range(10)]:
 [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]

```

Модуль random предоставляет функции для генерации случайных чисел, букв, случайного выбора элементов последовательности.

random.randint(A, B) - случайное целое число N, $A \leq N$

$\leq B$. `random.random()` - случайное число от 0 до 1.

Случайные числа в списке:

10 чисел, сгенерированных случайным образом в диапазоне
(10,80) `from random import randint`

```
l = [randint(10,80) for x in range(10)]
```

10 чисел, сгенерированных случайным образом в диапазоне
(0,1) l = [random() for i in range(10)]

```
from random import *
l = [randint(10,80) for i in range(10)]
print('10 чисел, сгенерированных случайным образом в диапазоне (10,80).')
print('l = [randint(10,80) for x in range(10)]:')
print(l)
print()

l = [random() for i in range(10)]
print('10 чисел сгенерированных в диапазоне от 0 до 1.')
print('l = [random() for i in range(10)]:')
for i in range(len(l)):
    print ('{: .2f}'.format(l[i]), end = " ")
```

```
10 чисел, сгенерированных случайным образом в диапазоне (10,80).
l = [randint(10,80) for x in range(10)]:
[70, 33, 79, 61, 34, 27, 11, 55, 52, 31]
```

```
10 чисел сгенерированных в диапазоне от 0 до 1.
l = [random() for i in range(10):
0.66 0.97 0.87 0.57 0.54 0.83 0.57 0.65 0.04 0.07
```

4. Ввод списка (массива) в языке Python.

Для ввода элементов списка используется цикл for и команда range

(): for i in range(N):

x[i] = int(input())

Более простой вариант ввода списка:

x = [int(input()) for i in range(N)]

```
print('Ввод списка. Пример 1:')
x=[]
for i in range(4):
    x.append(int(input()))
print(x)

x=[]
print('Ввод списка. Пример 2:')
x = [ int(input()) for i in range(4) ]
print(x)
```

```

Ввод списка. Пример 1:
45
4
85
2
[45, 4, 85, 2]
Ввод списка. Пример 2:
4
5
7
8
[4, 5, 7, 8]

```

Функция `int` здесь используется для того, чтобы строка, введенная пользователем, преобразовывалась в целые числа.

5. Вывод списка (массива) в языке

Python. Вывод целого списка

(массива):

```
print (L)
```

Поэлементный вывод списка (массива):

```
for i in range(N):
```

```
print ( L[i], end = " " )
```

```

...
Вывод целого списка (массива)
[1, 56, 6, 3, 6, 7, 3, 37, 7, 37, 37]

Поэлементный вывод списка (массива)
1 56 6 3 6 7 3 37 7 37 37

```

2. Методы списков.

Метод	Что делает
list.append(x)	Добавляет элемент в конец списка
list.extend(L)	Расширяет список list, добавляя в конец все элементы списка L
list.insert(i, x)	Вставляет перед i-ым элементом значение x
list.remove(x)	Удаляет первый элемент в списке, имеющий значение

	х. ValueError, если такого элемента не существует
list.pop([i])	Удаляет i-ый элемент и возвращает его. Если индекс не указан, удаляется последний элемент
list.index(x, [start [, end]])	Возвращает положение первого элемента со значением x (при этом поиск ведется от start до end)

list.count(x)	Возвращает количество элементов со значением x
list.reverse()	Разворачивает список
list.copy()	Поверхностная копия списка
list.clear()	Очищает список

Ниже приведена программа, демонстрирующая методы работы списков.

```

a=[0, 2, 2, 2, 4] #список a
b=[5, 6, 7, 2, 9] #список b
print ('Исходный список a:', a)
print ('Исходный список b:', b)
x=99
y=5

a.append(x)
print ('a.append(x):', a)

a.extend(b)
print ('a.extend(b):', a)

a.insert(3, x)
print ('a.insert(3, x):', a)

a.remove(x)
print ('a.remove(x):', a)

print ('a.pop(5):', a.pop(5))
print (a)

print ('a.index(y, 0, len(a)):', a.index(y, 0, len(a)))

print ('a.count(2):', a.count(2))

a.reverse()
print ('a.reverse():', a)

z=a.copy()
print ('z=a.copy():', z)

z.clear()
print ('z.clear():')
print ('z =', z)

```

Пример программы на Python

```

Исходный список a: [0, 2, 2, 2, 4]
Исходный список b: [5, 6, 7, 2, 9]
a.append(x): [0, 2, 2, 2, 4, 99]
a.extend(b): [0, 2, 2, 2, 4, 99, 5, 6, 7, 2, 9]
a.insert(3,x): [0, 2, 2, 99, 2, 4, 99, 5, 6, 7, 2, 9]
a.remove(x): [0, 2, 2, 2, 4, 99, 5, 6, 7, 2, 9]
a.pop(5): 99
[0, 2, 2, 2, 4, 5, 6, 7, 2, 9]
a.index(y,0,len(a)): 5
a.count(2): 4
a.reverse(): [9, 2, 7, 6, 5, 4, 2, 2, 2, 0]
z=a.copy(): [9, 2, 7, 6, 5, 4, 2, 2, 2, 0]
z.clear():
z = []

```

Результат выполнения программы

Вариант 0

1. Из массива X длиной n, среди элементов которого есть положительные, отрицательные и равные нулю, сформировать новый массив Y, взяв в него только те элементы из X, которые больше по модулю заданного числа M. Вывести на экран число M, данный и полученные массивы.

Решение:

```

n=int(input('Введите длину массива\n'))
m=int(input('Введите число M\n'))
x=[]
y=[]
for i in range(n):
    print('Введите ',i,'элемент:')
    x.append(int(input()))
for i in range(n):
    if abs(x[i])>m:
        y.append(x[i])
print('Введённое число M:',m)
print('Массив X:',x)
print('Массив Y:',y)

```

```

Введите длину массива
5
Введите число M
20
Введите 0 элемент:
21
Введите 1 элемент:
22
Введите 2 элемент:
5
Введите 3 элемент:
6
Введите 4 элемент:
8
Введённое число M: 20
Массив X: [21, 22, 5, 6, 8]
Массив Y: [21, 22]

```

2. В массиве целых чисел все отрицательные элементы заменить на положительные. Вывести исходный массив и полученный.

Решение:

```

n=int(input('Введите длину массива:'))
a=[]
for i in range(n):
    print('Введите',i,'элемент:')
    a.append(int(input()))
print('Исходный массив:',a)
for i in range(n):
    if a[i]<0:
        a[i]=-a[i]
print('Полученный массив:',a)

```

```

Введите длину массива:5
Введите 0 элемент:
-5
Введите 1 элемент:
-4
Введите 2 элемент:
-6
Введите 3 элемент:
5
Введите 4 элемент:
-7
Исходный массив: [-5, -4, -6, 5, -7]
Полученный массив: [5, 4, 6, 5, 7]

```

Вариант 1

1. Дан одномерный массив, состоящий из N целочисленных элементов. Ввести массив с клавиатуры. Найти максимальный элемент. Вывести массив на экран в обратном порядке.

2. В массиве действительных чисел все нулевые элементы заменить на среднее арифметическое всех элементов массива.

Вариант 2

1. Дан одномерный массив, состоящий из N целочисленных элементов. Ввести массив с клавиатуры. Найти минимальный элемент. Вывести индекс минимального элемента на экран.
2. Дан массив целых чисел. Переписать все положительные элементы во второй массив, а остальные - в третий.

Вариант 3

1. В одномерном числовом массиве D длиной n вычислить сумму элементов с нечетными индексами. Вывести на экран массив D , полученную сумму.
2. Дан одномерный массив из 8 элементов. Заменить все элементы массива меньшие 15 их удвоенными значениями. Вывести на экран монитора преобразованный массив.

Вариант 4

1. Дан массив целых чисел. Найти максимальный элемент массива и его порядковый номер.
2. Дан одномерный массив целого типа. Получить другой массив, состоящий только из нечетных чисел исходного массива или сообщить, что таких чисел нет. Полученный массив вывести в порядке убывания элементов.

Вариант 5

1. Дан одномерный массив из 10 целых чисел. Вывести пары отрицательных чисел, стоящих рядом.
2. Дан целочисленный массив размера 10. Создать новый массив, удалив все одинаковые элементы, оставив их 1 раз.

Вариант 6

1. Дан одномерный массив из 10 целых чисел. Найти максимальный элемент и сравнить с ним остальные элементы. Вывести количество меньших максимального и больших максимального элемента.
2. Одномерный массив из 10-и целых чисел заполнить с клавиатуры, определить сумму тех чисел, которые >5 .

Вариант 7

1. Дан массив целых чисел. Найти сумму элементов с четными номерами и произведение элементов с нечетными номерами. Вывести сумму и произведение.
2. Переставить в одномерном массиве минимальный элемент и максимальный.

Вариант 8

1. Найдите сумму и произведение элементов списка. Результаты вывести на экран.
2. В массиве действительных чисел все нулевые элементы заменить на среднее арифметическое всех элементов массива.

Вариант 9

1. Дан одномерный массив, состоящий из N вещественных элементов. Ввести массив с клавиатуры. Найти и вывести минимальный по модулю элемент. Вывести массив на экран в обратном порядке.
2. Даны массивы A и B одинакового размера 10. Вывести исходные массивы. Поменять местами их содержимое и вывести в начале элементы преобразованного массива A , а затем — элементы преобразованного массива B .

Вариант 10

- Определите, есть ли в списке повторяющиеся элементы, если да, то вывести на экран это значение, иначе сообщение об их отсутствии.
- Дан одномерный массив из 15 элементов. Элементам массива меньше 10 присвоить нулевые значения, а элементам больше 20 присвоить 1. Вывести на экран монитора первоначальный и преобразованный массивы в строчку.

Вариант 11

- 1 Найти наибольший элемент списка, который делится на 2 без остатка и вывести его на экран.
2. Дан одномерный массив целого типа. Получить другой массив, состоящий только из четных чисел исходного массива, меньше 10, или сообщить, что таких чисел нет. Полученный массив вывести в порядке возрастания элементов.

Вариант 12

- Найти наименьший нечетный элемент списка и вывести его на экран.
- Даны массивы А и В одинакового размера 10. Поменять местами их содержимое и вывести вначале элементы преобразованного массива А, а затем
 - элементы преобразованного массива В.

Вариант 13

1. Дан одномерный массив целых чисел. Проверить, есть ли в нем одинаковые элементы. Вывести эти элементы и их индексы.
2. Дан одномерный массив из 8 элементов. Заменить все элементы массива меньшие 15 их удвоенными значениями. Вывести на экран монитора преобразованный массив.

Вариант 14

1. Найти максимальный элемент численного массива и поменять его местами с минимальным.
2. Программа заполняет одномерный массив из 10 целых чисел числами, считанными с клавиатуры. Определить среднее арифметическое всех чисел массива. Заменить элементы массива большие среднего арифметического на 1.

Вариант 15

- Определите, есть ли в списке повторяющиеся элементы, если да, то вывести на экран эти значения.
- Дан одномерный массив целого типа. Получить другой массив, состоящий только из нечетных чисел исходного массива или сообщить, что таких чисел нет. Полученный массив вывести в порядке убывания элементов.

Практическое занятие 7 Функции и процедуры в Python

Цель работы: изучение процедур и функций в Python.

знать - синтаксис процедур и функций, процедура с параметром, локальные и глобальные переменные;

уметь - применять синтаксис процедур и функций при составлении программы; владеть - основными навыками работы с функциями и процедурами.

Подпрограмма - это именованный фрагмент программы, к которому можно обратиться из другого места программы

Подпрограммы делятся на две категории: процедуры и функции.

1. Процедуры.

Рассмотрим синтаксис процедуры:

```
def имя процедуры(Список параметров):
```

```
    Система команд
```

Для определения процедуры используется ключевое слово `def`, затем указывается имя процедуры и в скобках её формальные параметры, если они присутствуют. После ставится двоеточие и со следующей строки с отступом в 4 пробела указываются команды.

Процедура — вспомогательный алгоритм, выполняющий некоторые действия. Процедура должна быть определена к моменту её вызова. Определение процедуры начинается со служебного слова `def`.

Вызов процедуры осуществляется по её имени, за которым следуют круглые скобки, например, `Err()`.

В одной программе может быть сколько угодно много вызовов одной и той же процедуры.

Использование процедур сокращает код и повышает удобочитаемость.

Процедура с параметрами.

Как используются в Python параметры процедуры, рассмотрим на примере.

Пример.

Написать процедуру, которая печатает раз указанный символ (введенный с клавиатуры), каждый с новой строки.

```
def
printChar(s):
    print (s)
    sim = input('введите символ')

printChar(sim) # первый вызов, вывод введенного
символа printChar('*') # второй вызов, вывод *
```

```
def printChar(s):
    print (s)
    sim = input('введите символ: |')
    printChar(sim) # первый вызов, вывод введенного символа
    printChar('*') # второй вызов, вывод *
```

```
>>>
введите символ: 41
41
*
```

Глобальная переменная — если ей присвоено значение в основной программе (вне процедуры).

Локальная переменная (внутренняя) известна только на уровне процедуры, обратиться к ней из основной программы и из других процедур нельзя.

Параметры процедуры — локальные переменные.

2. Примеры использования локальных и глобальных переменных.

Пример 1.

```
x = 3 # глобальная переменная
def pr(): # процедура без параметров
    print (x) # вывод значения глобальной
    переменной pr()
```

```

x = 3 # глобальная переменная
def pr(): # процедура без параметров
    print (x) # вывод значения глобальной переменной
pr()

```

Пример 2.

`x = 3 # глобальная переменная`

`def pr(a): # процедура с
параметром print (a) # 4`

`pr(x) # передача параметра глобальной переменной (3)`

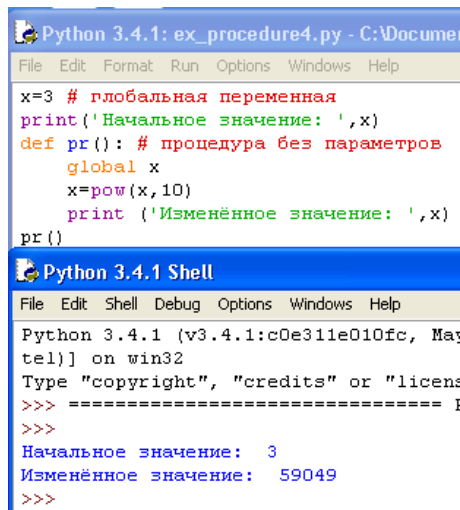
Существует возможность изменить значение глобальной переменной (не создавая локальную). В процедуре с помощью слова `global`:

`x = 3 # глобальная переменная`

`def pr(): # процедура без
параметров global x`

`x = pow(x,10)`

`print (x) # вывод измененного значения глобальной переменной pr()`



```
Python 3.4.1: ex_procedure4.py - C:\Docume
File Edit Format Run Options Windows Help
x=3 # глобальная переменная
print ('Начальное значение: ',x)
def pr(): # процедура без параметров
    global x
    x=pow(x,10)
    print ('Изменённое значение: ',x)
pr()

Python 3.4.1 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.1 (v3.4.1:c0e311e010fc, May
tel)] on win32
Type "copyright", "credits" or "licens
>>> =====
>>>
Начальное значение: 3
Изменённое значение: 59049
>>>
```

3. Функции.

Функция - подпрограмма, к которому можно обратиться из другого места программы. Для создания функции используется ключевое слово `def`, после которого указывается имя и список аргументов в круглых скобках. Тело функции выделяется также как тело условия (или цикла): четырьмя пробелами.

Рассмотрим синтаксис функции:

`def имя функции(Список параметров):`

Система

команд `return`

выражение

Часть функций языка Python являются встроенными функциями, которые обеспечены синтаксисом самого языка. Например, `int`, `input`, `randint`.

Рассмотрим пример создания пользовательских функций.

Пример 1.

Вычислить сумму цифр числа.

```
def sumD(n): # определение функции с
    параметром sumD = 0
```

while n!= 0:

sumD += n %

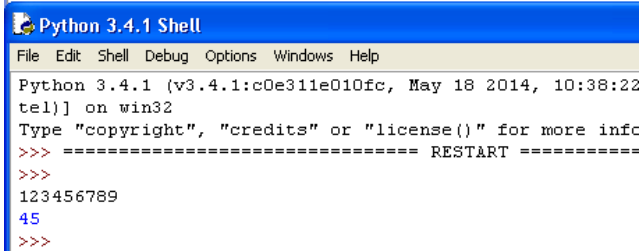
10 n = n // 10

return sumD # возврат значения

функции # основная программа

print (sumD(int(input()))) # вызов функции с параметром

```
def sumD(n): # определение функции с параметром
    summa = 0
    while n!= 0:
        summa += n % 10
        n=n//10
    return summa # возврат значения функции
# основная программа
print (sumD(int(input()))) # вызов функции с параметром
```



Python 3.4.1 Shell

File Edit Shell Debug Options Windows Help

Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 10:38:22
tel)] on win32
Type "copyright", "credits" or "license()" for more info
>>> ===== RESTART =====
>>>
123456789
45
>>>

Вариант 0.

1. Определить, являются ли три треугольника равновеликими. Длины сторон вводить с клавиатуры. Для подсчёта площади треугольника использовать формулу Герона. Вычисление площади оформить в виде функции с тремя параметрами.

Формула Герона:

$$S = \sqrt{p(p-a)(p-b)(p-c)},$$

$$\text{где } p = \frac{a+b+c}{2}$$

Решение:

```

import math
def s(x,y,z):
    p=(x+y+z)/2
    s=math.sqrt(p*(p-x)*(p-y)*(p-z))
    return s
A=[]
for i in range(3):
    print('Введите стороны ',i,'-го треугольника:')
    a=int(input('a:'))
    b=int(input('b:'))
    c=int(input('c:'))
    A.append(s(a,b,c))
for i in range(3):
    print('Площадь ',i,'-го треугольника {:.2f}'.format(A[i]))
if A[0]==A[1]:
    if A[0]==A[2]:
        print('Треугольники равновеликие')
else: print('Треугольники не равновеликие')

```

```

Введите стороны 0 -го треугольника:
a:3
b:4
c:5
Введите стороны 1 -го треугольника:
a:6
b:7
c:8
Введите стороны 2 -го треугольника:
a:9
b:10
c:11
Площадь 0 -го треугольника 6.00
Площадь 1 -го треугольника 20.33
Площадь 2 -го треугольника 42.43
Треугольники не равновеликие

```

2. Ввести одномерный массив A длиной m. Поменять в нём местами первый и последний элементы. Длину массива и его элементы ввести с клавиатуры. В программе описать процедуру для замены элементов массива. Вывести исходные и полученные массивы.

Решение:

```

def zam(X) :
    tmp=X[0]
    X[0]=X[len(X)-1]
    X[len(X)-1]=tmp
A=[]
m=int(input('Введите длину массива:'))
for i in range(m) :
    print('Введите ',i,'элемент массива')
    A.append(int(input()))
print(A)
zam(A)
print(A)

```

```

Введите длину массива:5
Введите 0 элемент массива
0
Введите 1 элемент массива
1
Введите 2 элемент массива
2
Введите 3 элемент массива
3
Введите 4 элемент массива
4
[0, 1, 2, 3, 4]
[4, 1, 2, 3, 0]

```

Вариант 1.

1. Составить программу для вычисления площади разных геометрических фигур.
2. Даны 3 различных массива целых чисел (размер каждого не превышает 15). В каждом массиве найти сумму элементов и среднеарифметическое значение.

Вариант 2.

1. Вычислить площадь правильного шестиугольника со стороной а, используя подпрограмму вычисления площади треугольника.
2. Пользователь вводит две стороны трех прямоугольников. Вывести их площади.

Вариант 3.

1. Даны катеты двух прямоугольных треугольников. Написать функцию вычисления длины гипотенузы этих треугольников. Сравнить и вывести какая из гипотенуз больше, а какая меньше.
2. Преобразовать строку так, чтобы буквы каждого слова в ней были отсортированы по алфавиту.

Вариант 4.

1. Даны две дроби A/B и C/D (A, B, C, D — натуральные числа). Составить программу деления дроби на дробь. Ответ должен быть несократимой дробью. Использовать подпрограмму алгоритма Евклида для определения НОД.
2. Задана окружность $(x-a)^2 + (y-b)^2 = R^2$ и точки $P(p1, p2)$, $F(f1, f1)$, $L(l1, l2)$. Выяснить и вывести на экран, сколько точек лежит внутри окружности. Проверку, лежит ли точка внутри окружности, оформить в виде процедуры.

Вариант 5.

- Даны две дроби A/B и C/D (A, B, C, D — натуральные числа). Составить программу вычитания из первой дроби второй. Ответ должен быть несократимой дробью. Использовать подпрограмму алгоритма Евклида для определения НОД.
- Напишите программу, которая выводит в одну строчку все делители переданного ей числа, разделяя их пробелами.

Вариант 6.

- Составить программу нахождения наибольшего общего делителя (НОД) и наименьшего общего кратного (НОК) двух натуральных чисел $НОК(A, B) = (A*B)/НОД(A,B)$. Использовать подпрограмму алгоритма Евклида для определения НОД.
- Составить программу вычисления площади выпуклого четырехугольника, заданного длинами четырех сторон и диагонали.

Вариант 7.

- Даны числа X, Y, Z, T — длины сторон четырехугольника.
Вычислить его

площадь, если угол между сторонами длиной X и Y — прямой. Использовать две подпрограммы для вычисления площадей: прямоугольного треугольника и прямоугольника.

– Напишите программу, которая переводит переданное ей неотрицательное целое число в 10-значный восьмеричный код, сохранив лидирующие нули.

Вариант 8.

- Найти все натуральные числа, не превосходящие заданного n , которые делятся на каждую из своих цифр. Ввести одномерный массив A длиной m . Поменять в нём местами первый и последний элементы. Длину массива и его элементы ввести с клавиатуры. В программе описать процедуру для замены элементов массива. Вывести исходные и полученные массивы.

Вариант 9.

1. Из заданного числа вычли сумму его цифр. Из результата вновь вычли сумму его цифр и т. д. Через сколько таких действий получится нуль?
2. Даны 3 различных массива целых чисел. В каждом массиве найти произведение элементов и среднеарифметическое значение.

Вариант 10.

1. На отрезке $[100, N]$ ($210 < N < 231$) найти количество чисел, составленных из цифр a, b, c .
2. Составить программу, которая изменяет последовательность слов в строке на обратную.

Вариант 11.

1. Два простых числа называются «близнецами», если они отличаются друг от друга на 2 (например, 41 и 43). Напечатать все пары «близнецов» из отрезка $[n, 2n]$, где n — заданное натуральное число, большее 2..
2. Даны две матрицы A и B . Написать программу, меняющую местами максимальные элементы этих матриц. Нахождение максимального элемента матрицы оформить в виде процедуры.

Вариант 12.

- Два натуральных числа называются «дружественными», если каждое из них

равно сумме всех делителей (кроме его самого) другого (например, числа 220 и 284). Найти все пары «дружественных» чисел, которые не больше данного числа N .

– Даны длины сторон треугольника a , b , c . Найти медианы треугольника, сторонами которого являются медианы исходного треугольника. Для вычисления медианы проведенной к стороне a , использовать формулу. Вычисление медианы оформить в виде процедуры.

Вариант 13.

4. Натуральное число, в записи которого n цифр, называется числом Армстронга, если сумма его цифр, возведенная в степень n , равна самому числу. Найти все числа Армстронга от 1 до k .

5. Три точки заданы своими координатами $X(x_1, x_2)$, $Y(y_1, y_2)$ и $Z(z_1, z_2)$. Найти и напечатать координаты точки, для которой угол между осью абсцисс и лучом, соединяющим начало координат с точкой, минимальный. Вычисления оформить в виде процедуры.

Вариант 14.

1. Составить программу для нахождения чисел из интервала $[M, N]$, имеющих наибольшее количество делителей.

2. Четыре точки заданы своими координатами $X(x_1, x_2)$, $Y(y_1, y_2)$, $Z(z_1, z_2)$, $P(p_1, p_2)$. Выяснить, какие из них находятся на максимальном расстоянии друг от друга и вывести на экран значение этого расстояния. Вычисление расстояния между двумя точками оформить в виде процедуры.

Вариант 15.

- Найти все простые натуральные числа, не превосходящие n , двоичная запись которых представляет собой палиндром, т. е. читается одинаково слева направо и справа налево.

- Четыре точки заданы своими координатами $X(x_1, x_2, x_3)$, $Y(y_1, y_2, y_3)$, $Z(z_1, z_2, z_3)$, $T(t_1, t_2, t_3)$. Выяснить, какие из них находятся на минимальном расстоянии друг от друга и вывести на экран значение этого расстояния. Вычисление расстояния между двумя точками оформить в виде процедуры.

Практическое занятие 8 Работа с двумерными массивами.

Цель работы: изучение двумерных массивов в Python.

знать - способ описания двумерного массива, способы ввода элементов двумерного массива;

уметь - вводить массивы, получать списки через присваивание конкретных значений, применять функции;

владеть - основными навыками создания программ обработки двумерных массивов.

Матрицами называются массивы элементов, представленные в виде прямоугольных таблиц, для которых определены правила математических действий. Элементами матрицы могут являться числа, алгебраические символы или математические функции.

Для работы с матрицами в Python также используются списки. Каждый элемент списка-матрицы содержит вложенный список.

Таким образом, получается структура из вложенных списков, количество которых определяет количество столбцов матрицы, а число элементов внутри каждого вложенного списка указывает на количество строк в исходной матрице.

1. Создание списка

Пусть даны два числа: количество строк n и количество столбцов m . Необходимо создать список размером $n \times m$, заполненный нулями.

Очевидное решение оказывается неверным:

$$A = [[0] * m] * n$$

В этом легко убедиться, если присвоить элементу $A[0][0]$ значение 1, а потом вывести значение другого элемента $A[1][0]$ — оно тоже будет равно 1! Дело в том, что $[0] * m$ возвращает ссылку на список из m нулей. Но последующее повторение этого элемента создает список из n элементов, которые являются ссылкой на один и тот же список (точно так же, как выполнение операции $B = A$ для списков не создает новый список), поэтому все строки результирующего списка на самом

деле являются одной и той же строкой.

Таким образом, двумерный список нельзя создавать при помощи операции повторения одной строки.

Первый способ.

Сначала создадим список из n элементов (для начала просто из n нулей). Затем сделаем каждый элемент списка ссылкой на другой одномерный список из m элементов:

```
A = [0] * n
```

```
    for i in
      range(n): A[i]
                = [0] * m
```

```
n=3
m=3
A=[0]*n
for i in range(n):
    A[i]=[0]*m
print('A:',A)
```

```
A: [[0, 0, 0], [0, 0, 0], [0, 0, 0]]
>>> |
```

Второй способ.

Создать пустой список, потом n раз добавить в него новый элемент, являющийся списком-строкой:

```
A = []
```

```
    for i in range(n):
      A.append([0] * m)
```

```
n=3
m=4
A = []
for i in range(n):
    A.append([0]*m)
print(A)
```

```
A: [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
>>> |
```

Ввод вложенного списка (двумерного массива)

Пример:

```
n=5
```

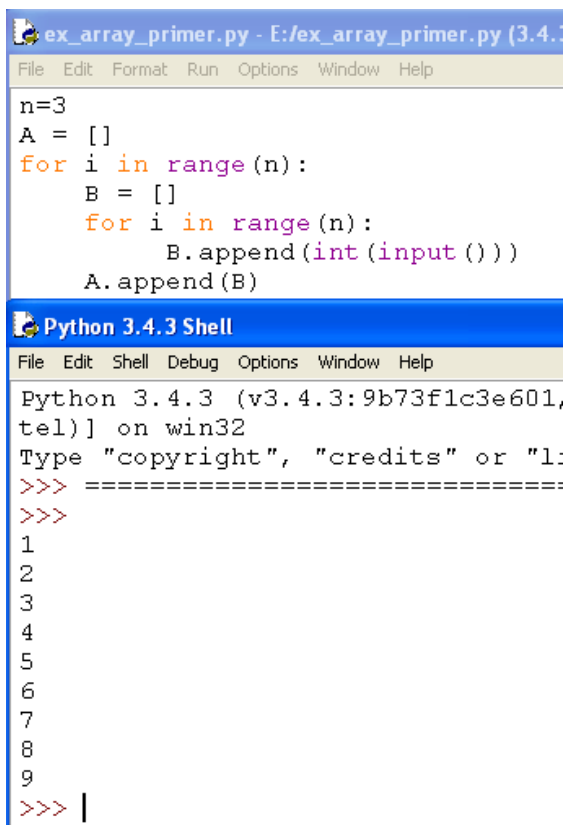
```
A =
```

```
  []
```

```
    for i in
      range(n): b =
```

```
input()
    for i in
        range(len(row)):
            row[i] = int(row[i])
```

A.append(row)



The image shows a screenshot of a Python IDE. The top window is titled 'ex_array_primer.py - E:/ex_array_primer.py (3.4.3)'. It contains the following code:

```
n=3
A = []
for i in range(n):
    B = []
    for i in range(n):
        B.append(int(input()))
    A.append(B)
```

The bottom window is titled 'Python 3.4.3 Shell'. It shows the output of the script:

```
Python 3.4.3 (v3.4.3:9b73f1c3e601,
tel)] on win32
Type "copyright", "credits" or "l:
>>> =====
>>>
1
2
3
4
5
6
7
8
9
>>> |
```

2. Вывод вложенного списка (двумерного массива)

Для обработки и вывода списка как правило используется два вложенных цикла. Первый цикл по номеру строки, второй цикл по элементам внутри строки. Например, вывести двумерный числовой список на экран построчно, разделяя числа пробелами внутри одной строки, можно так:

```
for i in range(n):
    for j in
    range(n):
        print(A[i][j], end = '
        ') print()
```



```

n=3
A = []
#ВВОД МАССИВА
for i in range(n):
    B = []
    for i in range(n):
        B.append(int(input()))
    A.append(B)
#ВЫВОД МАССИВА
for i in range(n):
    for j in range(n):
        print(A[i][j], end=' ')
    print()

```

```

1
2
3
4
5
6
7
8
9
1 2 3
4 5 6
7 8 9

```

То же самое, но циклы не по индексу, а по значениям списка:

```

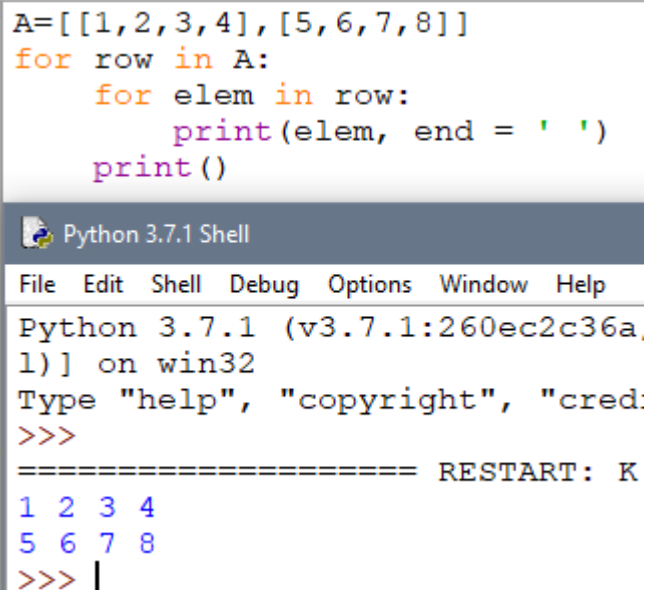
for row in A:
    for elem in row:
        print(elem, end = ' ')
    print()

```

```

A=[[1,2,3,4],[5,6,7,8]]
for row in A:
    for elem in row:
        print(elem, end = ' ')
    print()

```



Python 3.7.1 Shell

File Edit Shell Debug Options Window Help

Python 3.7.1 (v3.7.1:260ec2c36a, 1) on win32

Type "help", "copyright", "cred:

>>>

===== RESTART: K

```

1 2 3 4
5 6 7 8
>>> |

```

Для вывода одной строки можно воспользоваться методом `join`.
Используя этот метод в цикле `for`

можно `for row in A:`

```
print(' '.join(list(map(str, row))))
```

3. Обработка и вывод вложенных списков

Часто в задачах приходится хранить прямоугольные таблицы с данными. Такие таблицы называются матрицами или двумерными массивами. В языке программирования Python таблицу можно представить в виде списка строк, каждый элемент которого является в свою очередь списком, например, чисел. Например, создать числовую таблицу из двух строк и трех столбцов можно так:

```
A = [ [1, 2, 3], [4, 5, 6] ]
```

Здесь первая строка списка `A[0]` является списком из чисел `[1, 2, 3]`. То есть

```
A[0][0]= 1,
```

```
A[0][1]= 2,
```

```
A[0][2]= 3,
```

```
A[1][0]=4,
```

```
A[1][1]=5,
```

```
A[1][2]=6.
```

Используем два вложенных цикла для подсчета суммы всех чисел в списке:

```
S = 0
```

```
for i in range(len(A)):
```

```
    for j in
range(len(A[i])): S
    += A[i][j]
```

Или то же самое с циклом не по индексу, а по значениям строк:

```
S = 0
```

```
for row in A:
```

```
    for elem in
```

row:

S += elem

```
A=[[1, 2, 3,4],[ 5, 6,7,8]]
#вывод при помощи цикла for и метода join
print('Массив A:')
for i in A:
    print(' '.join(list(map(str, i))))
#Пример 1. Подсчёт суммы всех элементов
s = 0
for i in range(len(A)):
    for j in range(len(A[i])):
        s += A[i][j]
print('Пример 1. Сумма элементов:', s)
#Пример 2. Подсчёт суммы всех элементов
s = 0
for row in A:
    for elem in row:
        s += elem
print('Пример 2. Сумма элементов:', s)
```

```
Массив A:
1 2 3 4
5 6 7 8
Пример 1. Сумма элементов: 36
Пример 2. Сумма элементов: 36
```

4. Пример сложной обработки массива

Пусть дана квадратная матрица из n строк и n столбцов. Необходимо элементам, находящимся на главной диагонали, проходящей из левого верхнего угла в правый нижний (то есть тем элементам $A[i][j]$, для которых $i=j$) присвоить значение 0, элементам, находящимся выше главной диагонали – значение 1, элементам, находящимся ниже главной диагонали – значение 2. То есть получить такой массив (пример для $n=3$):

0 1 1

2 0 1

2 2 0

Рассмотрим несколько способов решения этой задачи.

Первый способ.

Элементы, которые лежат выше главной диагонали – это элементы $A[i][j]$, для которых $i < j$, а для элементов ниже главной диагонали $i > j$. Таким образом, мы можем сравнивать значения i и j и по ним определять значение $A[i][j]$. Получаем следующий алгоритм:

```
for i in range(n):
    for j in range(n):
        if i < j:
            A[i][j] = 0
        elif i > j:
            A[i][j] = 2
        else:
            A[i][j] = 1
```

Ниже приведён пример программы, в котором квадратная матрица 3×3 заполняется элементами со значением 9, а затем элементам, находящимся на главной диагонали, проходящей из левого верхнего угла в правый нижний (то есть тем элементам $A[i][j]$, для которых $i=j$) присваивается значение 0, элементам, находящимся выше главной диагонали – значение 1, элементам, находящимся ниже главной диагонали – значение 2.

```

n=3
A=[]
#заполняем массив 9-ми
for i in range(n):
    A.append([9]*n)
#вывод исходного массива
for i in range(n):
    for j in range(n):
        print(A[i][j], end = ' ')
    print()
#заменяем элементы главной диагонали, выше и ниже неё
for i in range(n):
    for j in range(n):
        if i < j:
            A[i][j] = 1
        elif i > j:
            A[i][j] = 2
        else:
            A[i][j] = 0
#вывод изменённого массива
print()
for i in range(n):
    for j in range(n):
        print(A[i][j], end = ' ')
    print()

```

```

9 9 9
9 9 9
9 9 9

0 1 1
2 0 1
2 2 0
>>> |

```

Второй способ.

Данный алгоритм плох, поскольку выполняет одну или две инструкции if для обработки каждого элемента. Если мы усложним алгоритм, то мы сможем обойтись вообще без условных инструкций.

Сначала заполним главную диагональ, для чего нам понадобится один цикл:

```

for i in
range(n): A[i]
[i] = 1

```

Затем заполним значением 0 все элементы выше главной диагонали, для чего нам понадобится в каждой из строк с номером i присвоить

значение элементам $A[i][j]$ для $j=i+1, \dots, n-1$. Здесь нам понадобятся вложенные циклы:

```
for i in range(n):  
    for j in range(i + 1,  
        n):  $A[i][j] = 0$ 
```

Аналогично присваиваем значение 2 элементам $A[i][j]$ для $j=0, \dots,$

```
        i-1: for i in range(n):  
            for j in range(0,  
                i):  $A[i][j] = 2$ 
```

Можно также внешние циклы объединить в один и получить еще одно, более компактное решение:

```
for i in range(n):  
    for j in range(0,  
        i):  $A[i][j] = 2$   
  
         $A[i][i] = 1$   
  
    for j in range(i + 1,  
        n):  $A[i][j] = 0$ 
```

```
n=3  
A=[]  
#заполняем массив 9-ми  
for i in range(n):  
    A.append([9]*n)  
#вывод исходного массива  
for i in range(n):  
    for j in range(n):  
        print(A[i][j], end = ' ')  
    print()  
#заменяем элементы главной диагонали, выше и ниже неё  
for i in range(n):  
    for j in range(0, i):  
        A[i][j] = 2  
    A[i][i] = 0  
    for j in range(i + 1, n):  
        A[i][j] = 1  
#вывод изменённого массива  
print()  
for i in range(n):  
    for j in range(n):  
        print(A[i][j], end = ' ')  
    print()
```

```

9 9 9
9 9 9
9 9 9

0 1 1
2 0 1
2 2 0
>>> |

```

Третий способ.

А вот такое решение использует операцию повторения списков для построения очередной строки списка. i -я строка списка состоит из i чисел 2, затем идет одно число 1, затем идет $n-i-1$ число 0:

```
for i in range(n):
```

$$A[i] = [2] * i + [1] + [0] * (n - i - 1)$$

```

n=3
A=[]
#заполняем массив 9-ми
for i in range(n):
    A.append([9]*n)
#вывод исходного массива
for i in range(n):
    for j in range(n):
        print(A[i][j], end = ' ')
    print()
#заменяем элементы главной диагонали, выше и ниже неё
for i in range(n):
    A[i] = [2] * i + [0] + [1] * (n - i - 1)
#вывод изменённого массива
print()
for i in range(n):
    for j in range(n):
        print(A[i][j], end = ' ')
    print()

```

```

9 9 9
9 9 9
9 9 9

0 1 1
2 0 1
2 2 0
>>> |

```


Вариант 0

- Дан двумерный массив размером

3x3. Определить максимальное значение среди элементов третьего столбца массива; максимальное значение среди элементов второй строки массива. Вывести полученные значения.

Решение:

```
n=3
a=[]
for i in range(n):
    b = []
    for j in range(n):
        print('Введите [',i,',',j,'] элемент')
        b.append(int(input()))
    a.append(b)
#вывод массива
for i in range(n):
    for j in range(n):
        print(a[i][j], end=' ')
    print()

#максимальное значение среди элементов третьего столбца
maximum=a[0][2]
for i in range(n):
    for j in range(n):
        if maximum<a[i][2]:
            maximum=a[i][2]
print('Максимальный в 3 столбце:',maximum)

#максимальное значение среди элементов второй строки
maximum=a[1][0]
for i in range(n):
    for j in range(n):
        if maximum<a[1][j]:
            maximum=a[1][j]
print('Максимальный во второй строке:',maximum)
```

```

Введите [ 0 , 0 ] элемент
1
Введите [ 0 , 1 ] элемент
2
Введите [ 0 , 2 ] элемент
3
Введите [ 1 , 0 ] элемент
4
Введите [ 1 , 1 ] элемент
5
Введите [ 1 , 2 ] элемент
6
Введите [ 2 , 0 ] элемент
7
Введите [ 2 , 1 ] элемент
8
Введите [ 2 , 2 ] элемент
9
1 2 3
4 5 6
7 8 9
Максимальный в 3 столбце: 9
Максимальный во второй строке: 6

```

- Дан двумерный массив размером $m \times n$. Сформировать новый массив заменив положительные элементы единицами, а отрицательные нулями. Вывести оба массива.

Решение:

```

m=int(input('Введите количество строк'))
n=int(input('Введите количество столбцов'))
a=[]
for i in range(m):
    b = []
    for j in range(n):
        print('Введите [',i,',',j,'] элемент')
        b.append(int(input()))
    a.append(b)
#вывод массива
print('Исходный массив:')
for i in range(m):
    for j in range(n):
        print(a[i][j], end=' ')
    print()

for i in range(m):
    for j in range(n):
        if a[i][j]<0: a[i][j]=0
        elif a[i][j]>0: a[i][j]=1

#вывод массива
print('Изменённый массив:')
for i in range(m):
    for j in range(n):
        print(a[i][j], end=' ')
    print()

```

```

Введите количество строк:3
Введите количество столбцов:4
Введите [ 0 , 0 ] элемент
-1
Введите [ 0 , 1 ] элемент
5
Введите [ 0 , 2 ] элемент
4
Введите [ 0 , 3 ] элемент
-5
Введите [ 1 , 0 ] элемент
-2
Введите [ 1 , 1 ] элемент
-1
Введите [ 1 , 2 ] элемент
0
Введите [ 1 , 3 ] элемент
4
Введите [ 2 , 0 ] элемент
-5
Введите [ 2 , 1 ] элемент
4
Введите [ 2 , 2 ] элемент
5
Введите [ 2 , 3 ] элемент
-5
Исходный массив:
-1 5 4 -5
-2 -1 0 4
-5 4 5 -5
Полученный массив:
0 1 1 0
0 0 0 1
0 1 1 0

```

Вариант 1.

1. Вычислить сумму и число положительных элементов матрицы $A[N, N]$, находящихся над главной диагональю.
2. Дана матрица $B[N, M]$. Найти в каждой строке матрицы максимальный и минимальный элементы и поменять их с первым и последним элементами строки соответственно.

Вариант 2.

- Дана целая квадратная матрица n -го порядка. Определить, является ли она магическим квадратом, т. е. такой матрицей, в которой суммы элементов во всех строках и столбцах одинаковы.
- Дана прямоугольная матрица $A[N, N]$. Переставить первый и последний столбцы местами и вывести на экран.

Вариант 3.

- I. Определить, является ли заданная целая квадратная матрица n -го порядка симметричной (относительно главной диагонали).

- II. Дана вещественная матрица размером $n \times m$. Переставляя ее строки и столбцы, добиться того, чтобы наибольший элемент (или один из них) оказался в верхнем левом углу.

Вариант 4.

1. Дана прямоугольная матрица. Найти строку с наибольшей и строку с наименьшей суммой элементов. Вывести на печать найденные строки и суммы их элементов.

2. Дана квадратная матрица $A[N, N]$, Записать на место отрицательных элементов матрицы нули, а на место положительных — единицы. Вывести на печать нижнюю треугольную матрицу в общепринятом виде.

Вариант 5.

1. Упорядочить по возрастанию элементы каждой строки матрицы размером $n \times m$.

2. Дана действительная матрица размером $n \times m$, все элементы которой различны. В каждой строке выбирается элемент с наименьшим значением. Если число четное, то заменяется нулем, нечетное - единицей. Вывести на экран новую матрицу.

Вариант 6.

3. Дана целочисленная квадратная матрица. Найти в каждой строке наибольший элемент и в каждом столбце наименьший. Вывести на экран.

- 4.

5. Дана действительная квадратная матрица порядка N (N — нечетное), все элементы которой различны. Найти наибольший элемент среди стоящих на главной и побочной диагоналях и поменять его местами с элементом, стоящим на пересечении этих диагоналей.

Вариант 7.

- Квадратная матрица, симметричная относительно главной диагонали, задана верхним треугольником в виде одномерного массива. Восстановить исходную матрицу и напечатать по строкам.

- Для заданной квадратной матрицы сформировать одномерный массив из ее диагональных элементов. Найти след матрицы, просуммировав элементы одномерного массива. Преобразовать исходную матрицу по правилу: четные строки разделить на полученное значение, нечетные оставить без изменения.

Вариант 8.

1. Задана матрица порядка n и число k . Разделить элементы k -й строки на диагональный элемент, расположенный в этой строке.
2. Задана квадратная матрица. Получить транспонированную матрицу (перевернутую относительно главной диагонали) и вывести на экран.

Вариант 9.

1. Для целочисленной квадратной матрицы найти число элементов, кратных k , и наибольший из этих элементов.
2. В данной действительной квадратной матрице порядка n найти наибольший по модулю элемент. Получить квадратную матрицу порядка $n - 1$ путем отбрасывания из исходной матрицы строки и столбца, на пересечении которых расположен элемент с найденным значением.

Вариант 10.

- a) Найти максимальный среди всех элементов тех строк заданной матрицы, которые упорядочены (либо по возрастанию, либо по убыванию).
- b) Расположить столбцы матрицы $D[M, N]$ в порядке возрастания элементов k -й строки ($1 \leq k \leq M$).

Вариант 11.

- a) В данной действительной квадратной матрице порядка n найти сумму элементов строки, в которой расположен элемент с наименьшим

значением. Предполагается, что такой элемент единственный.

в) Среди столбцов заданной целочисленной матрицы, содержащих только такие элементы, которые по модулю не больше 10, найти столбец с минимальным произведением элементов и поменять местами с соседним.

Вариант 12.

- Для заданной квадратной матрицы найти такие k , что k -я строка матрицы совпадает с k -м столбцом.
- Дана действительная матрица размером $n \times m$. Требуется преобразовать матрицу: поэлементно вычесть последнюю строку из всех строк, кроме последней.

Вариант 13.

- Определить наименьший элемент каждой четной строки матрицы $A[M, N]$.
- Найти наибольший и наименьший элементы прямоугольной матрицы и поменять их местами.

Вариант 14.

1. Задана квадратная матрица. Переставить строку с максимальным элементом на главной диагонали со строкой с заданным номером m .
2. Составить программу, которая заполняет квадратную матрицу порядка n натуральными числами $1, 2, 3, \dots, n^2$, записывая их в нее «по спирали».

Например, для $n = 5$ получаем следующую матрицу: 1 2 3 4
5

16 17 18 19 6

15 24 25 20 7

14 23 22 21 8

14 12 11 10 9

Вариант 15.

1. Определить номера строк матрицы $R[M, N]$, хотя бы один элемент которых равен c , и элементы этих строк умножить на d .
2. Среди тех строк целочисленной матрицы, которые содержат только нечетные элементы, найти строку с максимальной суммой модулей элементов.

Основные показатели оценки результата и их критерии

Критерии оценки

Практическая работа:

«5» – все задания выполнены

«4» – наблюдались неточности при выполнении работы

«3» – наблюдались ошибки при выполнении работы

«2» – работа выполнена менее 50 %

Тестовые задания:

Процент результативности (правильных ответов)	Качественная оценка индивидуальных образовательных достижений	
	балл (отметка)	вербальный аналог
86 ÷ 100	5	отлично

70 ÷ 85	4	хорошо
55 ÷ 69	3	удовлетворительно
менее 55	2	не удовлетворительно

Теоретические вопросы

Оценка	Критерии
<i>«отлично»</i>	Дан полный, развернутый ответ на поставленный вопрос, показана совокупность осознанных знаний по дисциплине, доказательно раскрыты основные положения вопросов; в ответе прослеживается четкая структура, логическая последовательность, отражающая сущность раскрываемых понятий, теорий, явлений. Знания по предмету демонстрируются на фоне понимания его в системе данной науки и междисциплинарных связей. Ответ изложен литературным языком с использованием современной технической терминологии. Могут быть допущены недочеты в определении понятий, исправленные студентом самостоятельно в процессе ответа.
<i>«хорошо»</i>	Дан полный, развернутый ответ на поставленный вопрос, показано умение выделить существенные и несущественные признаки, причинно-следственные связи. Ответ четко структурирован, логичен, изложен литературным языком с использованием современной технической терминологии. Могут быть допущены некоторые неточности или незначительные ошибки, исправленные студентом с помощью преподавателя.
<i>«удовлетворительно»</i>	Дан недостаточно полный и недостаточно развернутый ответ. Логика и последовательность изложения имеют нарушения. Допущены ошибки в раскрытии понятий, употреблении терминов. Студент не способен самостоятельно выделить существенные и несущественные признаки и причинно-следственные связи. В ответе отсутствуют выводы. Умение раскрыть значение обобщенных знаний не показано. Речевое оформление требует поправок, коррекции.

<p><i>«неудовлетворительно»</i></p>	<ol style="list-style-type: none">1. Ответ представляет собой разрозненные знания с существенными ошибками по вопросу. Присутствуют фрагментарность, нелогичность изложения. Студент не осознает связь обсуждаемого вопроса по билету с другими объектами дисциплины. Отсутствуют выводы, конкретизация и доказательность изложения. Речь неграмотная, техническая терминология не используется. Дополнительные и уточняющие вопросы преподавателя не приводят к коррекции ответа студента.2. Ответ на вопрос полностью отсутствует.3. Отказ от ответа.
-------------------------------------	---

Информационное обеспечение дисциплины

1. Лубашева, Т.В. Основы алгоритмизации и программирования: учебное пособие / Т.В. Лубашева, Б.А. Железко. - Минск: РИПО, 2016. - 378 с.: ил. - Библиогр. в кн. - ISBN 978-985-503-625-9; То же [Электронный ресурс]. - URL: <http://biblioclub.ru/index.php?page=book&id=463632>.
2. Коврижных, А.Ю. Основы алгоритмизации и программирования: практикум. В 2 ч. Ч. 2. Расчетные работы [Электронный ресурс]: учебно-методическое пособие / А.Ю. Коврижных, Е.А. Конончук, Г.Е. Лузина. — Электрон. дан. — Екатеринбург: УрФУ, 2016. — 44 с. — Режим доступа: <https://e.lanbook.com/book/98291>.
3. Коврижных, А.Ю. Основы алгоритмизации и программирования: практикум. В 2 ч. Ч. 1. Задачи и упражнения [Электронный ресурс]: учебно-методическое пособие / А.Ю. Коврижных, о.Е. К, Г.Е. Лузина. — Электрон. дан. — Екатеринбург: УрФУ, 2016. — 52 с. — Режим доступа: <https://e.lanbook.com/book/98290>.
4. *Златопольский Д.М.* Основы программирования на языке Python. – М.: ДМК Пресс, 2017. – 284 с.
5. Свейгарт, Эл. Автоматизация рутинных задач с помощью Python: практическое руководство для начинающих. Пер. с англ. — М.: Вильямс, 2016. – 592 с.