Лекция №4
РАБОТА СО СПИСКАМИ В ЯЗЫКЕ Python.

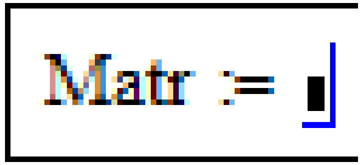
РАБОТА С МАТРИЦАМИ В ПАКЕТЕ MathCad. (продолжение)

ЗАДАЧИ ЛИНЕЙНОЙ АЛГЕБРЫ В СРЕДЕ ПАКЕТА MathCad.

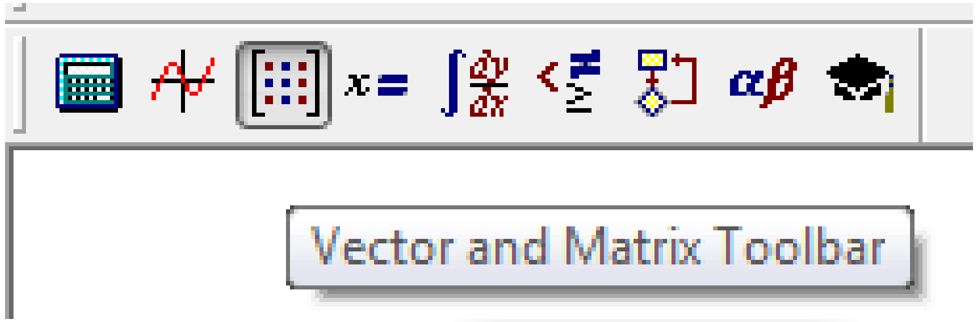
Определение и ввод матрицы в рабочий документ Mathcad

- Чтобы определить матрицу нужно:

1. ввести с клавиатуры имя матрицы и знак присваивания (либо нажав на клавиатуре комбинацию клавиш **<Shift>+<:>** или щелкнуть по кнопке<**:=**> панели **Evaluation**);

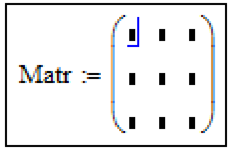


2. щелкнуть по кнопке **Vector or Matrix Toolbar**

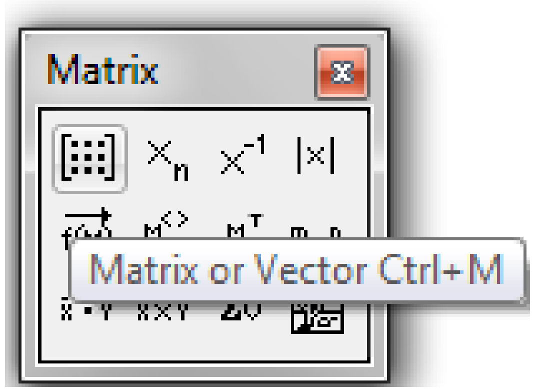


в панели математических инструме обы

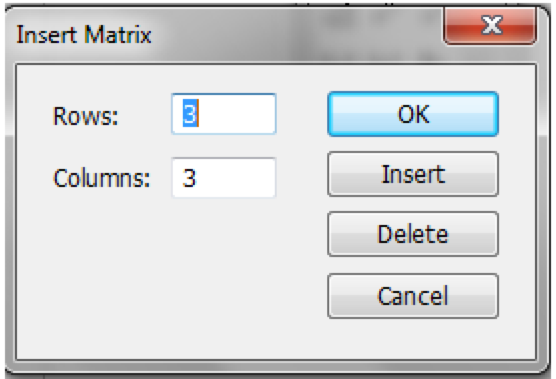
открыть панель матричных операц **ix**);



3. открыть щелчком по кнопке **Matrix or Vecto**r окно диалога определения размерности матрицы и ввести размерность матрицы: число строк (**Rows**), число столбцов (**Columns**);



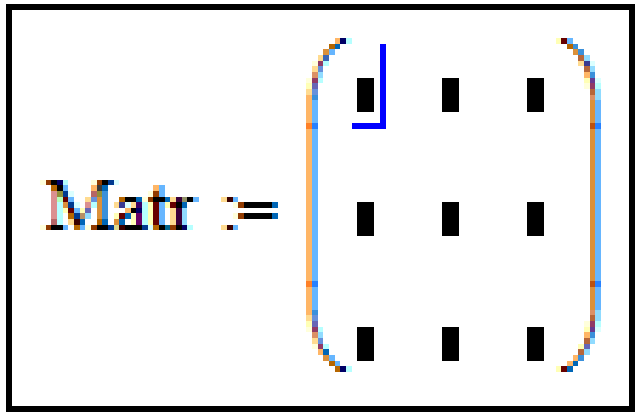
3. закрыть окно диалога, щелкнув по кнопке **Ok**.



ЗАДАЧИ ЛИНЕЙНОЙ АЛГЕБРЫ В СРЕДЕ ПАКЕТА MathCad.

Определение и ввод матрицы в рабочий документ Mathcad

- В рабочем документе, справа от знака присваивания, открывается поле ввода матрицы с помеченными позициями для ввода элементов. Для ввода элемента матрицы, установите курсор в помеченной позиции и введите с клавиатуры число или выражение.



НУМЕРАЦИЯ ЭЛЕМЕНТОВ МАТРИЦ И ВЕКТОРОВ

Номер первой строки (столбца) матрицы или первой компоненты вектора, хранится в Mathcad в переменной ORIGIN.

По умолчанию в Mathcad координаты векторов, столбцы и строки матрицы нумеруются начиная с 0 (ORIGIN:=0). Поскольку в математической записи чаще используется нумерация с 1, удобно перед началом работы с матрицами определять значение переменной ORIGIN равным 1, выполнять команду ORIGIN:=1.

ПАНЕЛЬ ОПЕРАЦИЙ С МАТРИЦАМИ И ВЕКТОРАМИ

Панель векторных и матричных операций открывается щелчком по кнопке **Vector and Matrix Toolbar** в панели математических инструментов. За кнопками панели закреплены следующие функции:

 — определение размеров матрицы;

 — ввод нижнего индекса;

 — вычисление обратной матрицы;

 — вычисление определителя матрицы: $|A|=\det A$; вычисление длины вектора $|x|$;

 — поэлементные операции с матрицами:

если $A=\{a_{ij}\}$, $B=\{b_{ij}\}$, то $\overline{AB}=\{a_{ij}b_{ij}\}$;

 — определение столбца матрицы: $M^{\langle j\rangle}$ — j-й столбец матрицы M;

 — транспонирование матрицы: $M=\{m_{ij}\}$, $M^T=\{m_{ji}\}$;

 — вычисление скалярного произведения векторов: $x*y=\sum_{i=1}^{n} x_i y_i$;

 — вычисление векторного произведения векторов:

$a \times b = (a_2 b_3 - a_3 b_2 , a_3 b_1 - a_1 b_3 , a_1 b_2 - a_2 b_1)$;

 — вычисление суммы компонент вектора: $\sum x=\sum_{i=1}^{n} x_i$;

 — определение диапазона изменения переменной;

 — визуализация цифровой информации, сохраненной в матрице.

Для того чтобы выполнить какую-либо операцию с помощью панели инструментов, нужно выделить матрицу и щелкнуть в панели по кнопке операции, либо щелкнуть по кнопке в панели и ввести в помеченной позиции имя матрицы.

ПАНЕЛЬ ОПЕРАЦИЙ С МАТРИЦАМИ И ВЕКТОРАМИ

Столбец 0

Столбец 2

Строка 0

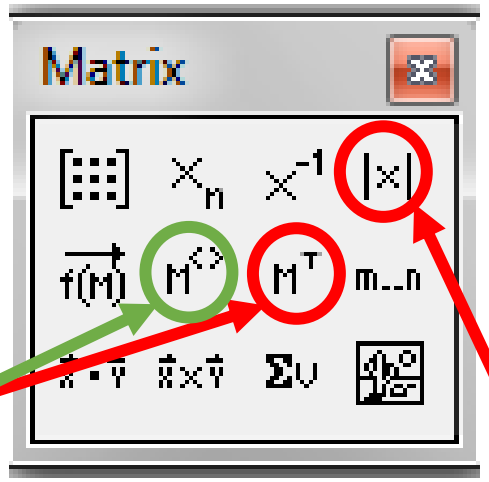
Строка 2

$$Matr := \begin{pmatrix} 1 & -2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

$Matr_{0,2} = 3$

$Transp := Matr^{T}$

$Matr_{2,0} = 7$



$Opred := |Matr| = -24$

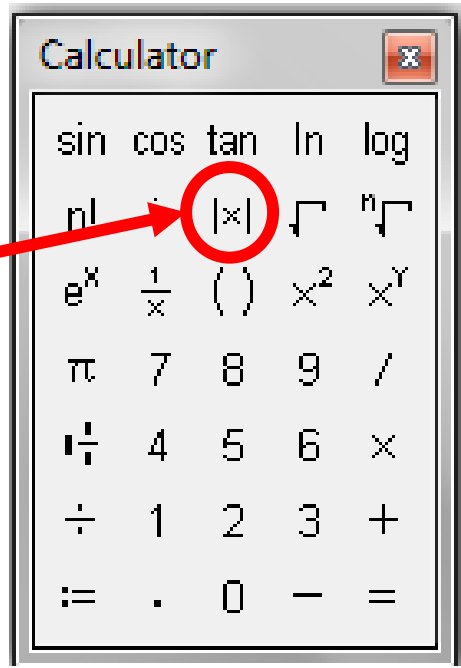
$$Vector1 := Matr^{\langle 0 \rangle} = \begin{pmatrix} 1 \\ 4 \\ 7 \end{pmatrix}$$

$$Vector2 := Matr^{\langle 1 \rangle} = \begin{pmatrix} -2 \\ 5 \\ 8 \end{pmatrix}$$

$$Transp = \begin{pmatrix} 1 & 4 & 7 \\ -2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix}$$

$Vector1_{0} = 1$

$Vector2_{2} = 8$



$dlina := |Vector1| = 8.124$

$$\sqrt{1^2 + 4^2 + 7^2} = 8.124$$

ОПЕРАЦИИ С МАТРИЦАМИ

$$Matr = \begin{pmatrix} 1 & -2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

$tmp := Matr^{\langle 0 \rangle}$

$Matr^{\langle 0 \rangle} := Matr^{\langle 1 \rangle}$

$Matr^{\langle 1 \rangle} := tmp$

Чему будет равна матрица Matr?

$$Matr = \begin{pmatrix} -2 & 1 & 3 \\ 5 & 4 & 6 \\ 8 & 7 & 9 \end{pmatrix}$$

Как поменять строки местами?

$$Matr = \begin{pmatrix} 1 & -2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

$Matr := Matr^{T}$

$tmp := Matr^{\langle 0 \rangle}$

$Matr^{\langle 0 \rangle} := Matr^{\langle 1 \rangle}$

$Matr^{\langle 1 \rangle} := tmp$

$Matr := Matr^{T}$

$$Matr = \begin{pmatrix} 4 & 5 & 6 \\ 1 & -2 & 3 \\ 7 & 8 & 9 \end{pmatrix}$$

МЕНЮ СИМВОЛЬНЫХ ОПЕРАЦИЙ С МАТРИЦАМИ

Меню символьных операций с матрицами (пункт Matrix меню Symbolics) содержит три функции:

- транспонирование (Transpose),
- обращение матрицы (Invert),
- вычисление определителя матрицы (Determinant).

Если требуется произвести какую-либо операцию через пункт Matrix меню Symbolics, нужно выделить матрицу и щелкнуть в меню по строке нужной операции.

Функции определения матриц и операции с блоками матриц

matrix(m, n, f) — создает и заполняет матрицу размерности **m** x **n**, элемент которой, расположенный в *i* -й строке, *j* -м столбце, равен значению *f(i, j)* функции *f(x, y)*;

$$ORIGIN := 1$$

Чтобы столбцы и строки матрицы нумеровались, начиная с единицы, присвойте переменной ORIGIN значение, равное единице.

$$f(x, y) := x + y$$

$$A := matrix(3, 4, f)$$

Создание и заполнение матрицы A размерности 3 x 4, элемент которой, расположенный в i -й строке, j -м столбце, равен значению f(i, j) функции f(x, y).

$$A = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 5 \end{pmatrix}$$

Для просмотра сформированной матрицы введите знак равенства, используя клавишу <=>.

Функции определения матриц и операции с блоками матриц

diag(v) — создает диагональную матрица, элементы главной диагонали которой хранятся в векторе **v**; **identity(n) —** создает единичную матрицу порядка **n**;

$i := 1..4 \quad v_i := i$

$B := diag(v)$ $\quad E := identity(3)$ $\quad A = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 5 \end{pmatrix}$

$D := augment(A, E)$ $\quad F := stack(A, B)$

$$B = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 4 \end{pmatrix} \quad E = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$F = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 5 \\ 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 4 \end{pmatrix}$$

$$D = \begin{pmatrix} 0 & 1 & 2 & 3 & 1 & 0 & 0 \\ 1 & 2 & 3 & 4 & 0 & 1 & 0 \\ 2 & 3 & 4 & 5 & 0 & 0 & 1 \end{pmatrix}$$

$G := submatrix(F, 4, 5, 1, 2) \quad G = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}$

augment(A, B) — формирует матрицу, в первых столбцах которой содержится матрица **A,** а в последних — матрица **B** (матрицы **A** и **B** имеют одинаковое число строк);

stack(A, B) — формирует матрицу, в первых строках которой содержится матрица **A,** а в последних — матрица **B** (матрицы **A** и **B** имеют оди наковое число столбцов);

submatrix(A, ir, jr, ic, jc) — формирует матрицу, которая является блоком матрицы **A**, расположенным в строках с **ir** по **jr** и в столбцах с **ic** по **jc**, **ir <= jr**, **ic <= jc**.

Функции отыскания различных числовых характеристик матриц

$i := 1..4 \qquad v_i := i \qquad B := \mathrm{diag}(v) \qquad B = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 4 \end{pmatrix}$

$E := \mathrm{identity}(3) \qquad E = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad A = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 5 \end{pmatrix}$

$D := \mathrm{augment}(A, E) \qquad D = \begin{pmatrix} 0 & 1 & 2 & 3 & 1 & 0 & 0 \\ 1 & 2 & 3 & 4 & 0 & 1 & 0 \\ 2 & 3 & 4 & 5 & 0 & 0 & 1 \end{pmatrix}$

$\mathrm{last}(v) = 4 \qquad \mathrm{length}(v) = 4$

$\mathrm{rows}(D) = 3 \qquad \mathrm{cols}(D) = 7 \qquad \max(D) = 5$

$\mathrm{tr}(B) = 10 \qquad \mathrm{rank}(A) = 2$

$\mathrm{norm1}(B) = 4 \qquad \mathrm{norm2}(B) = 4 \qquad \mathrm{normi}(B) = 4 \qquad \mathrm{norme}(B) = 5.477$

- **last(v)** — вычисление номера последнего элемента вектора **v**;
- **lenght(v)** — вычисление количества элементов **v** вектора;
- **rows(A)** — вычисление числа строк в матрице **A**;
- **cols(A)** — вычисление числа столбцов в матрице **A**;
- **max(A)** — вычисление наибольшего элемента в матрицы **A**;
- **tr(A)** — вычисление следа квадратной матрицы **A** (след матрицы равен сумме ее диагональных элементов);
- **rank(A)** — вычисление ранга матрицы **A**;
- **norm1(A), norm2(A), norme(A), normi(A)** — вычисление норм квадратной матрицы **A**.

Функции, реализующие численные алгоритмы решения задач линейной алгебры

$ORIGIN := 1$

Собственные значения матрицы C:

$$C := \begin{pmatrix} 1 & 2 & 3 \\ 0 & 1 & 2 \\ 3 & 4 & 1 \end{pmatrix} \qquad eigenvals(C) = \begin{pmatrix} -2.71 \\ 0.271 \\ 5.439 \end{pmatrix}$$

Собственные векторы, соответствующие собственным значениям матрицы C:

$$eigenvecs(C) = \begin{pmatrix} 0.415 & -0.76 & -0.625 \\ 0.432 & 0.611 & -0.321 \\ -0.801 & -0.222 & -0.711 \end{pmatrix}$$

Собственный вектор, соответствующий собственному значению L_2 матрицы C:

$$L := eigenvals(C) \qquad eigenvec(C, L_2) = \begin{pmatrix} -0.76 \\ 0.611 \\ -0.222 \end{pmatrix}$$

Решение системы линейных алгебраических уравнений Cx=b:

$$b := \begin{pmatrix} 14 \\ 8 \\ 14 \end{pmatrix} \qquad x := lsolve(C, b) \qquad x = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

rref(A) — приведение матрицы к ступенчатому виду с единичным базисным минором (выполняются элементарные операции со строками матрицы);

eigenvals(A) — вычисление собственных значений квадратной матрицы **A** ;

eigenvecs(A) — вычисление собственных векторов квадратной матрицы **A**; значением функции является матрица, столбцы которой есть собственные векторы матрицы **A**; порядок следования векторов отвечает порядку следования собственных значений, вычисленных функцией **eigenvals(A)**;

eigenvec(A, l) — вычисление собственного вектора матрицы **A,** отвечающего собственному значению **l**;

lsolve(A, b) — решение системы линейных алгебраических уравнений **Ax=b**.

Лекция №5
ФУНКЦИИ В PYTHON И СТРУКТУРНАЯ ПАРАДИГМА ПРОГРАММИРОВАНИЯ.

ЗАДАЧА

Даны два списка разной длины, вывести список, имеющий максимальную сумму.

Примечание: пользователь задает размерность списков, списки заполняются случайными вещественными числами.

Входные данные.

Размерность массивов length_of_first_array и length_of_second_array (целый тип данных).

Выходные данные.

Максимальная сумма (вещественный тип данных).

АЛГОРИТМ РЕШЕНИЯ ЗАДАЧИ

1. Считать входные данные: length_of_first_array и length_of_second_array (целый тип данных).
2. Создать два списка и заполнить каждый случайными числами.
3. Рассчитать суммы списков.
4. Сравнить суммы, полученные на третьем шаге и вывести максимальную сумму и элементы списка на экран.

ЛИСТИНГ ПРОГРАММНОЙ РЕАЛИЗАЦИИ ЗАДАЧИ

```
import sys
import random
try:
    length_of_first_array = int(input("Введите длину первого массива: "))
    length_of_second_array = int(input("Введите длину первого массива: "))
except ValueError:
    print("Вы ввели не целые числа")
    sys(1)

first_array = []
for i in range(length_of_first_array):
    first_array.append(round(random.uniform(-10, 10),2))

second_array = []
for i in range(length_of_second_array):
    second_array.append(round(random.uniform(-10, 10),2))
```

ЛИСТИНГ ПРОГРАММНОЙ РЕАЛИЗАЦИИ ЗАДАЧИ

```
print(first_array)
print(second_array)
first_sum = 0
for elem in first_array:
    first_sum += elem
second_sum = 0
for elem in second_array:
    second_sum += elem
if first_sum > second_sum:
    print(f"Сумма = {first_sum} Массив: {first_array}")
elif first_sum < second_sum:
    print(f"Сумма = {second_sum} Массив: {second_array}")
else:
    print(f"Сумма элементов двух массивов равны. Сумма = {first_sum}")
    print(f"Массивы: {first_array} и {second_array}")
```

```
Введите длину первого массива:
5
Введите длину первого массива:
3
[0.57, 6.06, -0.03, 6.47, -0.22]
[-8.28, -6.61, -3.53]
Сумма = 12.85 Массив: [0.57, 6.06, -0.03, 6.47, -0.22]


** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

ПРОЦЕДУРНОЕ ПРОГРАММИРОВАНИЕ В Python

Функции — именованная последовательность описаний и операторов, выполняющая какое-либо законченное действие. Функция может принимать параметры и возвращать значения.

Парадигма программирования, основанная на концепции вызова функций называется **процедурным программированием**.

Функции просто содержат ряд вычислительных шагов, которые необходимо выполнить.

Любая данная функция может быть вызвана в любой момент выполнения программы, в том числе другими функциями или самой собой, что позволяет повторно использовать один и тот же код в различных частях программ и на разных этапах его выполнения.

Например, функция print(), которая выводит некоторое значение на консоль. Python имеет множество встроенных функций и позволяет определять свои функции.

СИНТАКСИС ОПРЕДЕЛЕНИЯ ФУНКЦИИ БЕЗ ПАРАМЕТРОВ

```
1  def имя_функции ([параметры]) :
2      <инструкции>
```

Функция может возвращать результат. Для этого в функции используется оператор return, после которого указывается возвращаемое значение:

```
1  def имя_функции ([параметры]) :
2      <инструкции>
3      return <возвращаемое_значение>
```

Обратите внимание, что инструкции функции должны иметь отступы от начала функции. Если функция имеет одну инструкцию, то ее можно разместить на одной строке с остальным определением функции. Обычно между определением функции и остальными инструкциями, которые не входят в функцию, располагаются две пустых строки.

СИНТАКСИС ВЫЗОВА ФУНКЦИИ

Для вызова функции указывается имя функции, после которого в скобках идет передача значений для всех ее параметров. Функция должна быть объявлена до ее вызова.

Синтаксис вызова функции:

```
имя_функции ([параметры])
```

Например, определим и вызовем функции:

```
# Функция write_hallo без параметров, выводит на экран Hello
def write_hallo(): print("Hello")

def write(str):         # Функция write с однам входным параметром,
    print(str)          # выводит на экран значение входного параметра

write_hallo()           # Вызов функции write_hallo
write("Bye")            # Вызов функции write
```

ЛОКАЛЬНЫЕ ФУНКЦИИ

Одни функции могут определяться внутри других функций — внутренние функции еще называют локальными. Локальные функции можно использовать только внутри той функции, в которой они определены. Например:

```
1  def print_messages():
2      # определение локальных функций
3      def say_hello(str) : print(f"Hello {str}")
4      def say_goodbye() : print(f"Good Bye {str}")
5      str = input()
6      # вызов локальных функций
7      say_hello(str)
8      say_goodbye(str)
9  # Вызов функции print_messages
10 print_messages()
11 #say_hello() вне функции print_messages функция say_hello не доступна
```

ОРГАНИЗАЦИЯ ПРОГРАММЫ И ФУНКЦИЯ MAIN

Организация программы и функция main. В программе может быть определено множество функций. И чтобы всех их упорядочить, одним из способов их организации является добавление специальной функции (обычно называется main), в которой потом уже вызываются другие функции:

```
1  def main():
2      say_hello()
3      say_goodbye()
4
5   def say_hello():
6      print("Hello")
7   def say_goodbye():
8      print("Good Bye")
9   # Вызов функции main
10 main()
```

ЗНАЧЕНИЯ ПО УМОЛЧАНИЮ

Некоторые параметры функции мы можем сделать необязательными, указав для них значения по умолчанию при определении функции. Примечание: Если функция имеет несколько параметров, то необязательные параметры должны идти после обязательных. Например:

```
1  # функция возвращает сумму двух переданных в нее чисел
2  def sum(a = 0, b = 0):
3      return(a + b)
4
5  print(sum())
6  print(sum(2))
7  print(sum(2, 4))
```

ПЕРЕДАЧА ЗНАЧЕНИЙ ПАРАМЕТРАМ ПО ИМЕНИ

Именованные параметры. В примерах выше при вызове функции значения передаются параметрами функции по позиции. Но также можно передавать значения параметрам по имени. Для этого при вызове функции указывается имя параметра и ему присваивается значение:

```
1  def print_person(name, age):
2      print(f"Name: {name}  Age: {age}")
3
4  print_person(age = 22, name = "Tom")
```

ПЕРЕДАЧА ЗНАЧЕНИЙ ПАРАМЕТРАМ ПО ИМЕНИ

Символ « * » позволяет установить, какие параметры будут именнованными — то есть такие параметры, которым можно передать значения только по имени. Все параметры, которые располагаются справа от символа *, получают значения только по имени:

```
1  # параметры age и company являются именнованными.
2  def print_person(name, * , age, company):
3      print(f"Name: {name} Age: {age} Company: {company}")
4  print_person("Bob", age = 41, company ="Microsoft")
5  # Результат: Name: Bob Age: 41 company: Microsoft
6
```

ПЕРЕДАЧА ЗНАЧЕНИЙ ПАРАМЕТРАМ ПО ИМЕНИ

Можно сделать все параметры именнованными, поставив перед списком параметров символ « * ». Если наоборот надо определить параметры, которым можно передавать значения только по позиции, то есть позиционные параметры, то можно использовать символ « / » : все параметры, которые идут до символа « / » , являются позиционными и могут получать значения только по позиции:

```
1  # параметр name является позиционным
2  def print_person(name, /, age, company="Microsoft"):
3      print(f"Name: {name}  Age: {age}  Company: {company}")
4
5  print_person("Tom", company="JetBrains", age = 24)
6  # Результат: Name: Tom  Age: 24  company: JetBrains
7  print_person("Bob", 41)
8  # Результат: Name: Bob  Age: 41  company: Microsoft
```

ПЕРЕДАЧА ЗНАЧЕНИЙ ПАРАМЕТРАМ ПО ИМЕНИ

Для одной функции можно определять одновременно позиционные и именованные параметры:

```
''' параметр name располагается слева от символа /, поэтому является позиционным
и обязательным - ему можно можно передать значение только по позиции.
Параметр company является именованным, так как располагается справа от
символа *. Параметр age может получать значение по имени и по позиции. '''
def print_person(name, /,  age = 18, *, company):
    print(f"Name: {name}  Age: {age}  Company: {company}")

print_person("Sam", company ="Google")
# Результат: Name: Sam Age: 18 company: Google
print_person("Tom", 37, company ="JetBrains")
# Результат: Name: Tom Age: 37 company: JetBrains
print_person("Bob", company ="Microsoft", age = 42)
# Результат: Name: Bob Age: 42 company: Microsoft
```

ВОЗВРАЩЕНИЕ РЕЗУЛЬТАТА

Для возвращения результата из функции используется оператор **return**, после которого указывается возвращаемое значение. После **return** может идти сложное вычисляемое выражение, результат которого будет возвращаться из функции. **return** не только возвращает значение, но и производит выход из функции.

```
1  def sum(*numbers):
2      result = 0
3      for n in numbers:
4          result += n
5      return result
6      print(f"sum = {result}")
7  sum(1, 2, 3, 4, 5)          # На экран ничего выведено не будет, т.к. стр. 6
8   # стоит после оператора return, но функция возвращает результат
9  res = sum(3, 4, 5, 6)       # благодаря оператору в стр. 5
10 print(res)     # Результат: 18
```

ВОЗВРАЩЕНИЕ РЕЗУЛЬТАТА

Оператор **return** можно использовать без возвращающего значения. Типичная ситуация - в зависимости от определенных условий произвести выход из функции:

```
1  def print_person(name, age):
2      if age > 120 or age < 1:
3          print("Invalid age")
4          return
5      print(f"Name: {name}  Age: {age}")
6
7
8  print_person("Tom", 22)          # Результат: Name: Tom Age: 22")
9  print_person("Bob", -102)        # Результат: Invalid age
```

ФУНКЦИЯ КАК ТИП, ПАРАМЕТР И РЕЗУЛЬТАТ ДРУГОЙ ФУНКЦИИ

В Python **функция** фактически представляет **отдельный тип**, т.к. можно присвоить переменной какую-нибудь функцию и затем, **используя переменную, вызывать данную функцию**. Подобным образом можно через переменную вызывать функцию с параметрами и возвращать ее результат:

```
1  def sum(a, b): return a + b
2  def multiply(a, b): return a * b
3  operation = sum
4  result = operation(5, 6)
5  print(result)                # Результат: 11
6  operation = multiply
7  print(operation(5, 6))       # Результат: 30
8
```

ФУНКЦИЯ КАК ТИП, ПАРАМЕТР И РЕЗУЛЬТАТ ДРУГОЙ ФУНКЦИИ

Функцию можно передавать в качестве параметра в другую функцию. Таким образом, можно определять более гибкие по функциональности функции, которые через параметры принимают другие функции. Например, определим функцию, которая выводит на консоль значение функции y=f(x) на промежутке [a;b]:

```
1  import numpy
2  def print_func(a, b, h, func):
3      for x in numpy.arange(a, b+h, b) :
4          print(f"f( {i} ) = {func(i)} ")
5
6  def func1(x): return x*x+7
7  def func2(x): return 2+7*x
8
9  print_func(0, 5, 1, func1)
10 print_func(-2, 2, 0.5, func2)
```

ФУНКЦИЯ КАК ТИП, ПАРАМЕТР И РЕЗУЛЬТАТ ДРУГОЙ ФУНКЦИИ

```
1  def sum(a, b) : return a + b
2  def subtract(a, b) : return a - b
3  def multiply(a, b) : return a * b
4  def select_operation(choice):
5      if choice == "+":
6          return sum
7      elif choice == "-":
8          return subtract
9      else:
10         return multiply
11 operation = select_operation("-")      # operation = sum
12 print(operation(10, 6))                # Результат: 4
13 operation = select_operation("+")      # operation = sum
14 print(operation(10, 6))                # Результат: 16
```

Функция в Python может **возвращать другую функцию**, т.е. функция может возвращать ссылку на другую функцию. Например, определим функцию, которая в зависимости от значения параметра возвращает ту или иную операцию.

ЛИСТИНГ ПРОГРАММНОЙ РЕАЛИЗАЦИИ ЗАДАЧИ

```
1  import sys
2  import random
3  def create_array_of_random_float_numbers(length):
4      """Функция создает массив длиной length и
5      заполняет его случайными числами в диапозоне [-10;10]"""
6      array = []
7      for i in range(length):
8          array.append(round(random.uniform(-10, 10),2))
9      return array
10
11 def sum_elem_array(array)
12     """Функция возвразает сумму элементов массива array"""
13     sum = 0
14     for elem in array:
15         sum += elem
16     return sum
```

ЛИСТИНГ ПРОГРАММНОЙ РЕАЛИЗАЦИИ ЗАДАЧИ

```
17 try:
18     length_of_first_array = int(input("Введите длину первого массива: "))
19     length_of_second_array = int(input("Введите длину первого массива: "))
20 except ValueError:
21     print("Вы ввели не целые числа")
22     sys(1)
23
24 first_array = create_array_of_random_float_numbers(length_of_first_array)
25 second_array = create_array_of_random_float_numbers(length_of_second_array)
26
27 print(first_array)
28 print(second_array)
29
30 first_sum = sum_elem_array(first_array)
31 second_sum = sum_elem_array(second_array)
```

ЛИСТИНГ ПРОГРАММНОЙ РЕАЛИЗАЦИИ ЗАДАЧИ

```
32 if first_sum > second_sum:
33     print(f"Сумма = {first_sum} Массив: {first_array}")
34 elif first_sum < second_sum:
35     print(f"Сумма = {second_sum} Массив: {second_array}")
36 else:
37     print(f"Сумма элементов двух массивов равны. Сумма = {first_sum}")
40     print(f"Массивы: {first_array} и {second_array}")
```

```
Введите длину первого массива:
5
Введите длину первого массива:
3
[-9.22, 2.52, 4.29, -6.88, 6.21]
[-7.14, 3.44, -9.55]
Сумма = -3.080000000000001 Массив: [-9.22, 2.52, 4.29, -
6.88, 6.21]


** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

ОБЛАСТЬ ВИДИМОСТИ ПЕРЕМЕННЫХ

Область видимости переменных (**scope**) определяет **контекст переменной, в рамках которого ее можно использовать.** В Python есть два типа контекста: глобальный и локальный.

Глобальный контекст подразумевает, что **переменная является глобальной, она определена вне любой из функций и доступна любой функции в программе**.

В отличие от глобальных переменных **локальная переменная определяется внутри функции и доступна только из этой функции, то есть имеет локальную область видимости.**

Примечание: локальная переменная скрывают глобальную с тем же именем.

ОБЛАСТЬ ВИДИМОСТИ ПЕРЕМЕННЫХ

Пример:

```
1   i = 5           # определение глобальной переменной i
2   def func(a):
3       i = a       # определение локальной переменной i, скрывающей
4                   # глобальную переменную с тем же именем
5       b = 100
6       print(i)    # вывод значения локальной переменной i на консоль
7   print(i)        # вывод значения глобальной переменной i на консоль.
8                   # Результат 5
9   func(3)         # Вызов функции. Результат: 3
10  print(i)        # значение глобальной переменной не изменилось.
11                  # Результат: 5
12  # print(b)      # переменная b — локальная, область ее видимости —
13  тело # функции func()
```

ОБЛАСТЬ ВИДИМОСТИ ПЕРЕМЕННЫХ

Если же мы хотим **изменить в локальной функции глобальную переменную**, а не определить локальную, то необходимо использовать ключевое слово **global**:

```
1  i = 5            # определение глобальной переменной I
2  def func(a):
3      global i     # благодаря ключевому слову global присваиваем
4      i = a        # глобальной переменной значение переменной a
5      print(i)     # вывод значения локальной переменной i на консоль
6
7  print(i)         # Результат: 5
8  func(3)          # Результат: 3
9  print(i)         # значение глобальной переменной изменилось.
   Результат:3
```

ОБЛАСТЬ ВИДИМОСТИ ПЕРЕМЕННЫХ

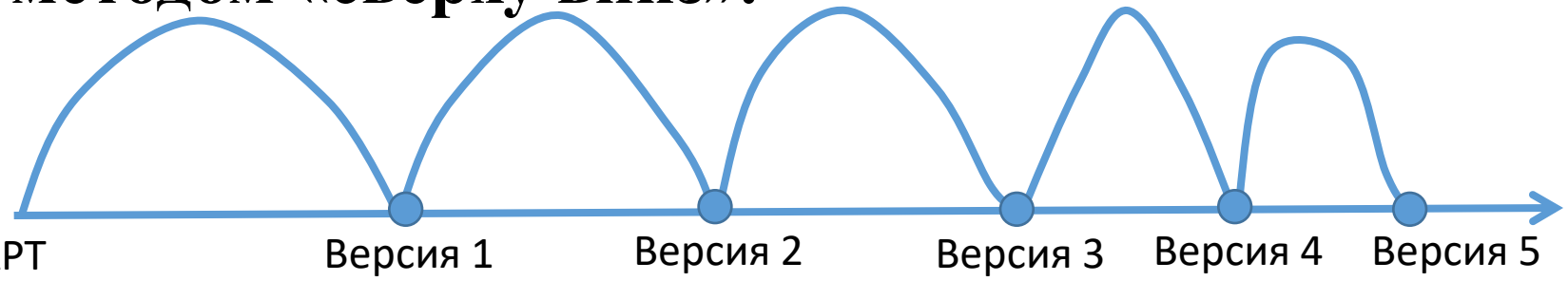
Выражение **nonlocal** прикрепляет идентификатор к переменной из ближайшего окружающего контекста (за исключением глобального контекста). Обычно nonlocal **применяется во вложенных функциях**, когда надо прикрепить идентификатор за переменной или параметром окружающей внешней функции. Выражение nonlocal может **пригодиться для того, чтобы во вложенной функции указать, что идентификатор во вложенной функции будет представлять переменную из окружающей функции:**

```
1  def outer():              # внешняя функция outer()
2      n = 5
3      def inner():          # вложенная функция
4          nonlocal n        # указываем, что n - это переменная из окружающей  функции inner()
5          n = 25
6          print(n)
7      inner()               # Результат: 25
8      print(n)
9  outer()                   # Результат: 25
```

СТРУКТУРНАЯ ПАРАДИГМА ПРОГРАММИРОВАНИЯ

Структу́рное программи́рование — парадигма программирования, в основе которой лежит представление программы в виде иерархической структуры блоков. Концептуализирована в конце 1960-х — начале 1970-х годов на фундаменте теоремы Бёма — Якопини, математически обосновывающей возможность структурной организации программ, и работы Эдсгера Дейкстры «О вреде оператора goto» (англ. Goto considered harmful).

В соответствии с парадигмой, любая программа, которая строится без использования **оператора goto**, состоит из трёх базовых управляющих конструкций: последовательность, ветвление, цикл; кроме того, используются подпрограммы. **При этом разработка программы ведётся пошагово, методом «сверху вниз».**



СТРУКТУРНАЯ ПАРАДИГМА ПРОГРАММИРОВАНИЯ

Структурное программирование призвано, в частности, устранить беспорядок и ошибки в программах, вызванные трудностями чтения кода, несистематизированным, неудобным для восприятия и анализа исходным текстом программы. Такой текст нередко характеризуют как «спагетти-код».

Спагетти-код — плохо спроектированная, слабо структурированная, запутанная и трудная для понимания программа, содержащая много операторов goto (особенно переходов назад), исключений и других конструкций, ухудшающих структурированность[8]. Один из самых известных антипаттернов программирования.

Спагетти-код назван так потому, что ход выполнения программы похож на миску спагетти, то есть извилистый и запутанный. Иногда называется «кенгуру-код» (kangaroo code) из-за множества инструкций jump.

Спагетти-код может быть отлажен и работать правильно и с высокой производительностью, но он крайне сложен в сопровождении и развитии[8]. Доработка спагетти-кода для добавления новой функциональности иногда несет значительный потенциал внесения новых ошибок. По этой причине становится практически неизбежным рефакторинг — главное лекарство от спагетти.

ПРОГРАММИРОВАНИЕ «СВЕРХУ ВНИЗ»

Программа – совокупность фрагментов, которые, принимая некоторые данные, вырабатывают результат и передают его следующему фрагменту.

Алгоритм разбиения задачи на фрагменты:

1) проанализировать задачу и выделить в ней фрагменты;

2) отобразить процесс разбиения в виде блок-схемы или линейной схемы и пронумеровать в ней фрагменты;

3) установить между выделенными фрагментами связи: для каждого фрагмента определить, какие данные он получает (входные данные) и какие данные возвращает (выходные данные). **Связи между фрагментами называются интерфейсом;**

4) рассмотреть далее каждый фрагмент самостоятельно; разработать для него алгоритм и записать его либо в виде линейной схемы, либо в виде блок-схемы. При необходимости подвергнуть фрагмент разбиению на более мелкие фрагменты. Такое разбиение продолжать до тех пор, пока не будут получены фрагменты, программирование которых не составляет особых затруднений;

5) оформить выделенные фрагменты в виде программных компонентов.

ВЕРСИЯ 1

```
1  import sys
2  import random
3  def main():
4      try:
5          length_of_first_array = int(input("Введите длину первого массива: "))
6          length_of_second_array = int(input("Введите длину первого массива:"))
7      except ValueError:
8          print("Вы ввели не целые числа")
9          sys(1)
10     compare(length_of_first_array,length_of_second_array)
11
12 def compare(length_of_first_array,length_of_second_array):
13     """Функция выполняет поставленную задачу"""
14     pass
15
16 main()
```

ВЕРСИЯ 2

```
1  import sys
2  import random
3  def main():
4      try:
5          length_of_first_array = int(input("Введите длину первого массива: "))
6          length_of_second_array = int(input("Введите длину первого массива:
7  "))
8      except ValueError:
9          print("Вы ввели не целые числа")
10         sys(1)
11     compare(length_of_first_array,length_of_second_array)
12
13 def compare(length_of_first_array,length_of_second_array):
14     """Функция выполняет поставленную задачу"""
15     first_array = create_array_of_random_float_numbers(length_of_first_array)
16     second_array =
```

ВЕРСИЯ 3

```
1  import sys
2  import random
3  def main():
4      try:
5          length_of_first_array = int(input("Введите длину первого массива: "))
6          length_of_second_array = int(input("Введите длину первого массива:
7  "))
8      except ValueError:
9          print("Вы ввели не целые числа")
10         sys(1)
11     compare(length_of_first_array,length_of_second_array)
12
13 def compare(length_of_first_array,length_of_second_array):
14     """Функция выполняет поставленную задачу"""
15     first_array = create_array_of_random_float_numbers(length_of_first_array)
16     second_array =
```