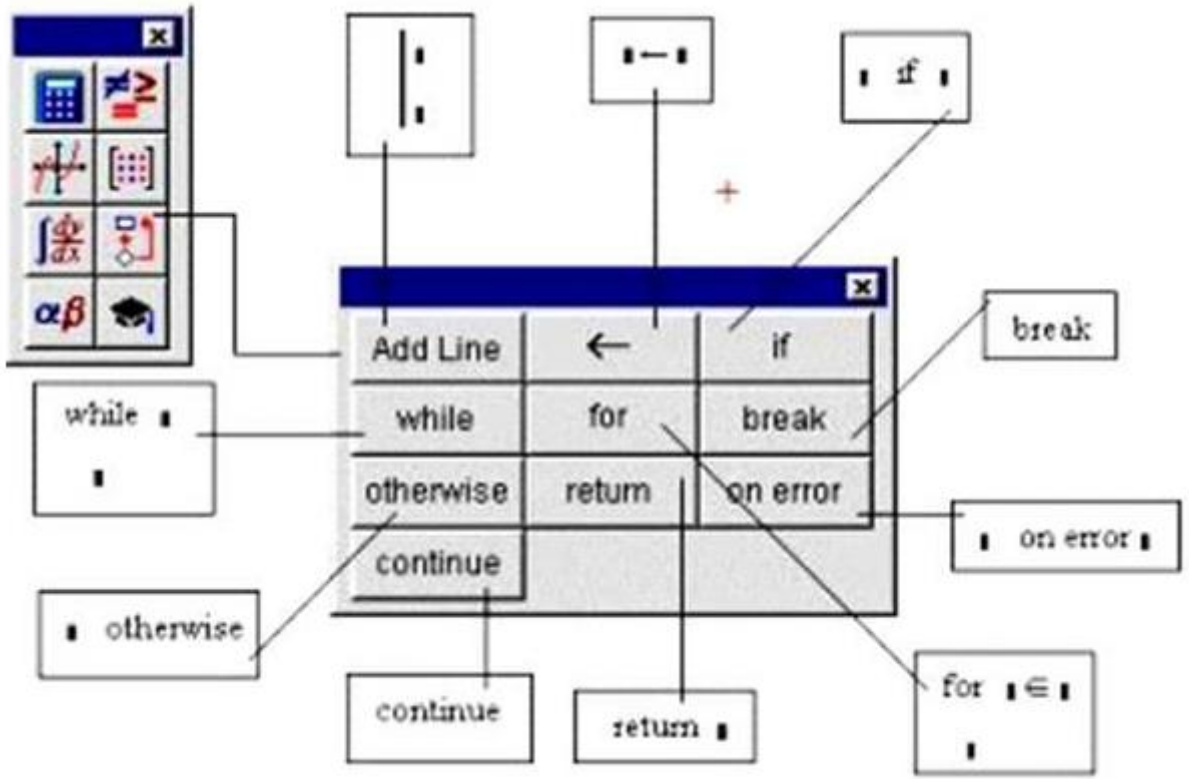
# Лекция №3 (Продолжение) Ветвление и циклы в Python. Модульное программирование в пакете MathCad.

# МОДУЛЬНОЕ ПРОГРАММИРОВАНИЕ В ПАКЕТЕ MathCAD

Основные операторы программирования расположены на панели расположены на панели **Программирование (Programming Toolbar)**.

Описание подпрограммы-функции (П-Ф) размещается в рабочем документе перед ее вызовом
и включает в себя:

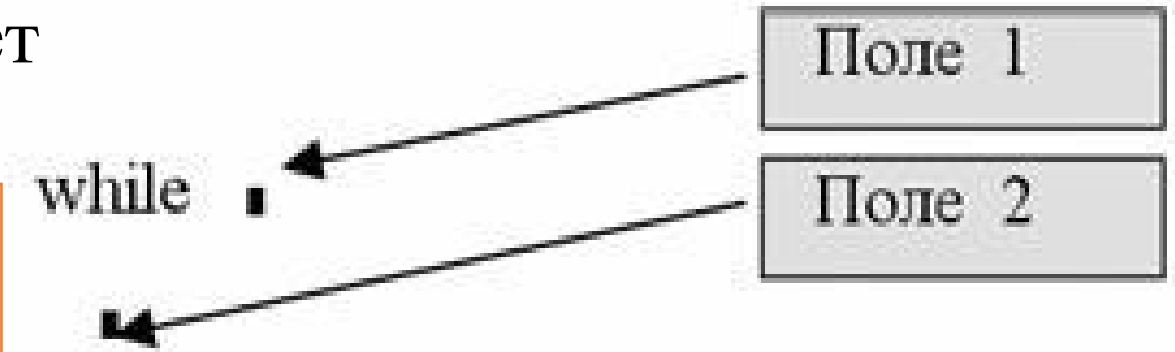
- имя подпрограммы-функции,
- список формальных параметров (который может отсутствовать),
- тело подпрограммы-функции.



# МОДУЛЬНОЕ ПРОГРАММИРОВАНИЕ В ПАКЕТЕ MathCAD. ЦИКЛ С ПРЕДУСЛОВИЕМ

Для программирования таких циклов используется оператор цикла **while**. Для ввода этого оператора необходимо выполнить следующие действия:

1. щелкнуть на кнопке while палитры ПРОГРАММИРОВАНИЕ. На экране появляются поля ввода;
2. в поле 1 ввести условие выполнения цикла;
3. в поле 2 ввести операторы тела цикла. В теле цикла должны присутствовать операторы, которые могут изменить значение условия цикла, иначе цикл будет продолжаться бесконечно.




Оператор цикла while выполняется следующим образом: обнаружив оператор while, MathCAD проверяет указанное в операторе условие. Если оно равно 1 (т.е. выполняется), то выполняется тело цикла, и снова проверяется условие. Если условие принимает значение 0, то цикл заканчивается.

# ПРОГРАММИРОВАНИЕ ЦИКЛИЧЕСКИХ АЛГОРИТМОВ

**Пример.** Программа печатает таблицу значений функции $\mathbf{y=x^2+1}$ во введенном диапазоне.

**Входные данные**:

*xn* (вещественное число) – начало диапазона.

*xk* (вещественное число) – конец диапазона.

*dx* (вещественное число) – шаг изменения аргумента.

**Выходные данные:**

$y(xn), y(xn+dx), y(xn+2dx),\ldots, y(xk)$, (вещественные числа).

$$\text{func}(x_n, x_k, dx) := \left| \begin{array}{l} \text{return error("Первый > Второго")} \text{ if } x_n > x_k \\ x \leftarrow x_n \\ i \leftarrow 0 \\ \text{while } x \le x_k \\ \quad \left| \begin{array}{l} y_i \leftarrow x^2 + 1 \\ i \leftarrow i + 1 \\ x \leftarrow x + dx \end{array} \right. \\ y \end{array} \right.$$

$$\text{res} := \text{func}(1, 3, 1) = \begin{pmatrix} 2 \\ 5 \\ 10 \end{pmatrix}$$

res := func(3 , 1 , 1) =

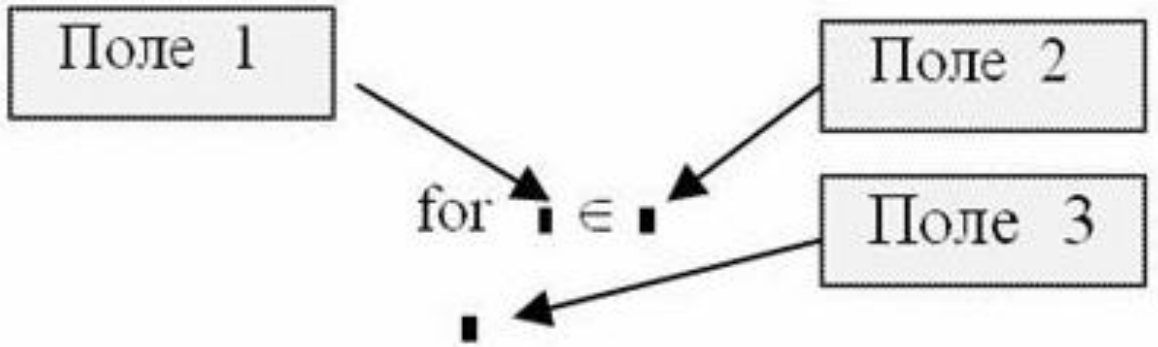
Первый > Второго

```
1  import math
2  xn = float(input(" Начало диапазона: "))
3  xk = float(input(" Конец  диапазона: "))
4  dx = float(input(" Шаг: "))
5  x = xn
6  while x <= xk :
7      y = math.pow(x, 2) + 1
8      print("y(",x,") = ", y)
9      x += dx
10 else :
11     if xn > xk :
12         print("Значение начала диапазона
13 > значения конца диапазона")
```

# МОДУЛЬНОЕ ПРОГРАММИРОВАНИЕ В ПАКЕТЕ MathCAD. ЦИКЛ С ПАРАМЕТРОМ

Для ввода такого оператора необходимо выполнить следующие действия:

1. щелкнуть на кнопке **for** палитры ПРОГРАММИРОВАНИЕ. На экране появятся поля ввода;
2. в поле ввода 1 ввести имя переменной, являющейся параметром цикла;
3. в поле 2 — закон изменения параметра цикла, используя для этого описание дискретной переменной или описание массива;
4. в поле 3 — операторы, составляющие тело цикла. Если одной строки недостаточно, то дополнительные поля ввода (дополнительные строки) создаются щелчком на кнопке Add line палитры ПРОГРАММИРОВАНИЕ, и тогда слева от тела цикла появляется вертикальная черта, охватывающая тело цикла.




Оператор for предоставляет возможность организовать цикл по некоторой переменной, изменяющейся в заданном диапазоне.

**for** <переменная> Є <начальное значение>, <шаг> .. <конечное значение> оператор

Эта запись означает, что оператор выполнится для значений переменной, изменяющейся в диапазоне от начального значения до конечного, с заданным шагом.

$$\text{func}(x_n, x_k, dx) := \left| \begin{array}{l} \text{return } \text{error}(\text{"Первый} > \text{Второго"}) \text{ if } x_n > x_k \\ i \leftarrow 0 \\ \text{for } x \in x_n, x_n + dx .. x_k \\ \quad \left| \begin{array}{l} y_i \leftarrow x^2 + 1 \\ i \leftarrow i + 1 \end{array} \right. \\ y \end{array} \right.$$

$$\text{res} := \text{func}(1, 3, 1) = \begin{pmatrix} 2 \\ 5 \\ 10 \end{pmatrix}$$

```
1  import math
2  import numpy
3  xn = float(input(" Начало диапазона: "))
4  xk = float(input(" Конец  диапазона: "))
5  dx = float(input(" Шаг: "))
6  for x in numpy.arange(xn, xk+dx, dx) :
7      y = math.pow(x, 2) + 1
8      print("y(",x,") = ", y)
9  else :
10     if xn > xk :
11         print("Значение начала диапазона
12 > значения конца диапазона")
```

НГТУ НЭТИ

# МОДУЛЬНОЕ ПРОГРАММИРОВАНИЕ В ПАКЕТЕ MathCAD. ЦИКЛ С ПРЕДУСЛОВИЕМ

К сожалению, организация итерационного цикла с помощью оператора while без дополнительных средств контроля может привести к зацикливанию, т.е. повторению тела цикла «бесконечное» число раз.

Например, если вы забудете про увеличение параметра цикла. Поэтому в MathCAD имеется специальный оператор break, который позволяет выйти из цикла или приостановить исполнение программы при выполнении заданного в операторе break условия.

Оператор break используется в левом поле ввода условного оператор if, а в правом размещается условие, при выполнении которого происходит прекращение работы цикла или программы. Поэтому первоначально вводится оператор if, а затем заполняются поля этого оператора.

# МОДУЛЬНОЕ ПРОГРАММИРОВАНИЕ В ПАКЕТЕ MathCAD. ЦИКЛ С ПРЕДУСЛОВИЕМ

Пример. Составим П-Ф, реализующую итерационную процедуру вычисления корня квадратного без «зацикливания». Пример показывает написание подпрограммы без «зацикливания» с использованием оператора break.

$$sqroot\_new(9, 0.0001) = \begin{pmatrix} 3 \\ 0 \end{pmatrix}$$

$$sqroot\_new(9, -0.0001) = \begin{pmatrix} "not\ solve" \\ 1 \end{pmatrix}$$

$$sqroot\_new(a, \varepsilon) :=$$
$$xn \leftarrow a$$
$$ierr \leftarrow 1$$
$$\text{for } i \in 1..10000$$
$$\quad \text{if } |xn^2 - a| < \varepsilon$$
$$\quad\quad ierr \leftarrow 0$$
$$\quad\quad \text{break}$$
$$\quad xc \leftarrow xn$$
$$\quad xn \leftarrow \frac{xc + \frac{a}{xc}}{2}$$
$$xn \leftarrow "not\ solve" \quad \text{if } ierr = 1$$
$$\begin{pmatrix} xn \\ ierr \end{pmatrix}$$

# МОДУЛЬНОЕ ПРОГРАММИРОВАНИЕ В ПАКЕТЕ MathCAD. ЦИКЛ С ПАРАМЕТРОМ

Пример. Составить П-Ф, осуществляющую суммирование ряда с бесконечным числом слагаемых. Накопление суммы прекращается, как только очередное слагаемое по абсолютной величине становится меньше заданной погрешности.

Заметим, что вторым формальным параметром является имя функции пользователя, определяющей зависимость величины члена ряда от его номера. При вызове этот формальный параметр заменяется фактическим – именем функции пользователя, описанной до обращения к П-Ф.

$$b(i) := \frac{1}{i^2}$$

$$d(i) := \frac{1}{\sqrt{i}}$$

```
sum_conv(ε, a) :=
    sum ← 0
    ierr ← 1
    for i ∈ 1..10000
        if |a(i)| < ε
            ierr ← 0
            break
        sum ← sum + a(i)
    sum ← "not convergence"  if ierr = 1
    (sum)
    (ierr)
```

$$sum\_conv(0.00001, b) = \begin{pmatrix} 1.64177 \\ 0 \end{pmatrix}$$

$$sum\_conv(0.00001, d) = \begin{pmatrix} \text{"not convergence"} \\ 1 \end{pmatrix}$$

НГТУ НЭТИ

# Лекция №4
# РАБОТА СО СПИСКАМИ В ЯЗЫКЕ Python.

# РАБОТА С МАТРИЦАМИ В ПАКЕТЕ MathCad.

# ОСНОВЫ РАБОТЫ СО СПИСКАМИ В Python

**Список** (list) представляет **тип данных, который хранит набор или последовательность элементов**. Во многих языках программирования есть аналогичная структура данных, которая называется массив. Однако, в отличии от массива, **элементами** списка могут быть значения **разных типов**.

Способы создания списка. **Для создания списка применяются квадратные скобки []**, внутри которых через запятую перечисляются элементы списка. Также для создания списка можно использовать **функцию-конструктор list()**.

# ПРИМЕР СОЗДАНИЯ СПИСКОВ

```
1  numbers1 = []      # numbers1 хранит пустой список
2  numbers2 = list() # numbers2 хранит пустой список
3  # numbers хранит список, элементами которого являются числа
4  numbers = [1, 2, 3, 4, 5]
5  # people хранит список, элементами которого являются строки
6  people = ["Tom", "Sam", "Bob"]
7  # objects хранит список, элем-ми которого разные по типу данных
8  objects = [1, 2.6, "Hello", True]
9  print(objects)              # Результат: [1 2.6 "Hello" True]
```

Как видно из примера, для проверки элементов списка можно использовать стандартную функцию print.

# ПРИМЕР СОЗДАНИЯ СПИСКОВ

Конструктор list может принимать набор значений, на основе которых создается список:

```
1  numbers1 = [1, 2, 3, 4, 5]
2  numbers2 = list(numbers1) # в numbers2 скопирован список из numbers1
3  print(numbers2)           # Результат: [1, 2, 3, 4, 5]
4  letters = list("Hello")
5  print(letters)            # Результат: ['H', 'e', 'l', 'l', 'o']
```

# ПРИМЕР СОЗДАНИЯ СПИСКОВ

Для создания списка, в котором повторяется одно и то же значение несколько раз, можно использовать символ звездочки — «*» (фактически применить операцию умножения к уже существующему списку):

```
1  numbers = [5] * 6                    # 6 раз повторяем 5
2  print(numbers)                       # Результат: [5, 5, 5, 5, 5, 5]
3  people = ["Tom"] * 3                 # 3 раза повторяем "Tom"
4  print(people)                        # Результат: ["Tom", "Tom", "Tom"]
5  students = ["Bob", "Sam"] * 2        # 2 раза повторяем "Bob", "Sam"
6  print(students)                      # ["Bob", "Sam", "Bob", "Sam"]
```

**ОБРАЩЕНИЕ К ЭЛЕМЕНТАМ СПИСКА**

осуществляется при помощи индексации (по номеру элемента в списка). Индексы начинаются с нуля (первый элемент иметь индекс 0, второй элемент - индекс 1 и так далее). Для обращения к элементам с конца можно использовать отрицательные индексы, начиная с -1 (последний элемент имеет индекс -1, предпоследний - -2 и так далее):

```
1  import numpy as np                    # подключение библиотеки numpy
2  # объявление и инициализация списка с именем mas
3  Mas = np.arange( 1, 10, 2 )                 # mas = [1, 3, 5, 7, 9]
4  mas[2] = 100                                # mas = [1, 3, 100, 7, 9]
5  for i in range( 0, len( mas ) – 1 ) : # при помощи цикла выводим на экран
6      print( mas[ i ], end = ", " )           # все элементы, кроме последнего
7  print( mas[ -1 ], end = "." )               # выводим последний элемент
8  # Результат: 1, 3, 100, 7, 9.
```

# РАЗЛОЖЕНИЕ СПИСКА НА ОТДЕЛЬНЫЕ ЭЛЕМЕНТЫ

Python позволяет разложить список на отдельные элементы, однако следует учитывать, что количество переменных должно быть равно числу элементов присваиваемого списка:

```
1  mas = [1, 2, 3]
2  a, b, c = mas
3  print(f"a = {a}, b = {b}, c =
4  {c}")
5  # Результат: a = 1, b = 2, c = 3
```

```
1  mas = [1, 2, 3]
2  a, b = mas
3  print(f"a = {a}, b = {b}")
4  '''в стоке 2 ошибка ValueError:
5  too many values to unpack '''
```

# ПЕРЕБОР ЭЛЕМЕНТОВ СПИСКА

Для перебора элементов можно использовать как **цикл for**, так и **цикл while**:

```
1   mas = [1, 3, 5]
2   sum = 0
3   for elem in mas :          # цикл для подсчета суммы элементов списка
4       sum += elem
5   product = 1
6   i = 0
7   while i < len(mas) : # цикл для подсчета произведения элементов списка
8       product *= mas[ i ]    # применяем индекс для получения элемента
9       i += 1                 # ОБЯЗАТЕЛЬНО. Увеличиваем счетчик цикла
10  print(f"Сумма = { sum }, Произведение = { product }.")
11  # Результат: Сумма = 9, Произведение = 15.
```

# СРАВНЕНИЕ СПИСКОВ

Python позволяет сравнивать списки. Два списка считаются равными, если они содержат один и тот же набор элементов:

```
1  numbers1 = [1, 2, 3, 4, 5]
2  numbers2 = list( [1, 2, 3, 4, 5] )
3  if numbers1 == numbers2 :                    # Сравнение двух списков
4      print("numbers1 equal to numbers2")
5  else:
6      ("numbers1 is not equal to numbers2")
7  # Результат: numbers1 equal to numbers2.
```

# МЕТОДЫ И ФУНКЦИИ ПО РАБОТЕ СО СПИСКАМИ

Для управления элементами списки имеют целый ряд **методов**. Некоторые из них:

- **append(item)**: добавляет элемент item в конец списка;
- **insert(index, item)**: добавляет элемент item в список по индексу index;
- **extend(items)**: добавляет набор элементов items в конец списка
- **remove(item)**: удаляет элемент item. Удаляется только первое вхождение элемента. Если элемент не найден, генерирует исключение ValueError;
- **clear()**: удаление всех элементов из списка;
- **index(item)**: возвращает индекс элемента item. Если элемент не найден, генерирует исключение ValueError;
- **pop([index])**: удаляет и возвращает элемент по индексу index. Если индекс не передан, то просто удаляет последний элемент;

# МЕТОДЫ И ФУНКЦИИ ПО РАБОТЕ СО СПИСКАМИ

- **count(item)**: возвращает количество вхождений элемента item в список;
- **sort([key])**: сортирует элементы. По умолчанию сортирует по возрастанию. Но с помощью параметра key мы можем передать функцию сортировки;
- **reverse()**: расставляет все элементы в списке в обратном порядке;
- **copy()**: копирует список;

Кроме того, Python предоставляет ряд встроенных **функций** для работы со списками:

- **len(list)**: возвращает длину списка;
- **sorted(list, [key])**: возвращает отсортированный список;
- **min(list)**: возвращает наименьший элемент списка;
- **max(list)**: возвращает наибольший элемент списка.

# ДОБАВЛЕНИЕ И УДАЛЕНИЕ ЭЛЕМЕНТОВ

```
1   mas = [1, 2]
2   mas.append(3)          # добавляем 3 в конец списка: mas = [1, 2, 3]
3   mas.insert(1, 4)       # добавляем 4 на вторую позицию: mas = [1, 4, 2, 3]
4   mas.extend([8, 9])     # добавляем набор элементов в конец списка: mas = [1, 4, 2, 3, 8, 9]
5   i = 0
6   while i < len(mas) : # в цикле удалям все четные элементы из списка
7       if mas[i] % 2 == 0 :
8           mas.pop(i)
9       else:
10          i += 1
11  # После цикла mas = [1, 3, 9]
12  if (len(mas)):          # если список не пустой, то
13      mas.pop()  # удаляем последний элемент: mas = [1, 3]
14  delete_elem = 1
```

# ДОБАВЛЕНИЕ И УДАЛЕНИЕ ЭЛЕМЕНТОВ

```
15  # Проверяем, есть ли в списке элемент со значением,
16  # хранящимся в переменной delete_elem
17  if delete_elem in mas :
18      index = mas.index(delete_elem) # если есть, получаем индекс элемента
19      mas.pop(index)    # и удаляем элемент по этому индексу: mas = [3]
20  print(mas)                        # Результат: [3]
21  mas.clear()                       # удаляем все элементы
22  print(mas)                        # Результат: []
```

*Примечание*: прежде чем удалить элемент с определенным значение, необходимо проверить его наличие в списке (13-14 и 17-19 строчки примера). Т.к. если определенный элемент не найден, то методы remove и index генерируют исключение.

# УДАЛЕНИЕ ЭЛЕМЕНТОВ ИЗ СПИСКА

Python также поддерживает еще один способ удаления элементов списка — с помощью оператора del. В качестве параметра этому оператору передается удаляемый элемент или набор элементов:

```
1  mas = [10, 15, 17, 100, 8, 4]
2  del mas[1]          # удаляем второй элемент
3  print(mas)          # Результат: mas = [10, 17, 100, 8, 4]
4  del mas[:3]         # удаляем  по четвертый элемент не включая
5  print(mas)          # Результат: mas = [8, 4]
6  del mas[1:]         # удаляем  со второго элемента
7  print(mas)          # Результат: mas = [8]
```

## ПОДСЧЕТ ВХОЖДЕНИЙ

Если необходимо узнать, сколько раз в списке присутствует тот или иной элемент, то можно применить метод count():

```
1  mas = [1, 2, 3, 3, 2, 2]
2  print(mas.count(2))      # Результат: 3
```

## СОРТИРОВКА ЭЛЕМЕНТОВ СПИСКА

Метод sort() сортирует список по возрастанию, метод reverse() — по убыванию:

```
1  mas = [10, -2, 3, 7, 1]
2  mas.sort()
3  print(mas)          # Результат: [-2, 1, 3, 7, 10]
4  mas.reverse()
5  print(mas)          # Результат: [10, 7, 3, 1, -2]
```

# СОРТИРОВКА

```
1  people = ["Tom", "bob", "alice", "Sam", "Bill"]
2  people.sort()                  # стандартная сортировка
3  print(people)        # Результат: ["Bill", "Sam", "Tom", "alice", "bob"]
4  people.sort(key=str.lower)     # сортировка без учета регистра
5  print(people)        # Результат: ["alice", "Bill", "bob", "Sam", "Tom"]
```

При сортировке фактически сравниваются два объекта, и который из них "меньше", ставится перед тем, который "больше". Понятия "больше" и "меньше" довольно условны. И если для чисел все просто — числа расставляются в порядке возрастания, то для строк и других объектов ситуация сложнее. В частности, строки оцениваются по первым символам. Если первые символы равны, оцениваются вторые символы и так далее. При чем цифровой символ считается "меньше", чем алфавитный заглавный символ, а заглавный символ считается меньше, чем строчный.

Таким образом, если в списке сочетаются строки с верхним и нижним регистром, то мы можем получить не совсем корректные результаты, так как для нас строка "bob" должна стоять до строки "Tom". И чтобы изменить стандартное поведение сортировки, мы можем передать в метод sort() в качестве параметра функцию:

# СОРТИРОВКА С УСЛОВИЕМ

Кроме метода sort мы можем использовать встроенную функцию **sorted**, синтаксис которой имеет вид:

**sorted(list, key): сортирует список list, применяя к элементам функцию key**

При использовании этой функции следует учитывать, что эта функция не изменяет сортируемый список, а все отсортированные элементы она помещает в новый список, который возвращается в качестве результата:

```
1  people = ["Tom", "bob", "alice", "Sam", "Bill"]
2  sorted_people = sorted(people, key=str.lower)
3  print(sorted_people) # Результат: ["alice", "Bill", "bob", "Sam", "Tom"]
```

# МИНИМАЛЬНОЕ И МАКСИМАЛЬНОЕ ЗНАЧЕНИЯ

Встроенный функции Python **min()** и **max()** позволяют найти минимальное и максимальное значения соответственно:

```
1 mas = [9, 21, 12, 1, 3, 15, 18]
2 print(min(mas))          # Результат: 1
3 print(max(mas))          # Результат: 21
```

# КОПИРОВАНИЕ СПИСКОВ

При копировании списков следует учитывать, что **списки представляют изменяемый (mutable) тип, поэтому если обе переменных будут указывать на один и тот же список, то изменение одной переменной, затронет и другую переменную**:

```
1  array1 = [1, 2, 3]
2  array2 = array1
3  array2.append(4) # добавляем элемент во второй список
4  # array1 и array2 указывают на один и тот же список
5  print(array1)       # Результат: [1, 2, 3, 4]
6  print(array2)       # Результат: [1, 2, 3, 4]
```

Это так называемое "**поверхностное копирование**" (shallow copy). И, как правило, такое поведение нежелательное.

# КОПИРОВАНИЕ СПИСКОВ

Чтобы происходило копирование элементов, но при этом переменные указывали на разные списки, необходимо выполнить **глубокое копирование** (deep copy). Для этого можно использовать **метод copy()**:

```
1  array1 = [1, 2, 3]
2  array2 = array1.copy()
3  array2.append(4) # добавляем элемент во второй список
4  # array1 и array2 указывают на один и тот же список
5  print(array1)       # Результат: [1, 2, 3]
6  print(array2)       # Результат: [1, 2, 3, 4]
```

# КОПИРОВАНИЕ ЧАСТИ СПИСКА

Если необходимо скопировать не весь список, а только его какую-то определенную часть, то мы можем применять специальный синтаксис. который может принимать следующие формы:

**list[:end]**: через параметр end передается индекс элемента, до которого нужно копировать список

**list[start:end]**: параметр start указывает на индекс элемента, начиная с которого надо скопировать элементы

**list[start:end:step]**: параметр step указывает на шаг, через который будут копироваться элементы из списка. По умолчанию этот параметр равен 1.

# СОЕДИНЕНИЕ СПИСКОВ

Для объединения списков применяется **операция сложения (+).**

```
1  array1 = [0, 1, 2, 3, 4, 5, 6, 7]
2  array2 = array1[:3].copy()
3  print(array2)                    # Результат: [0, 1, 2]
4  array3 = array1[2:5].copy()
5  print(array3)                    # Результат: [2, 3, 4]
6  array4 = array1[1:6:2].copy()
7  print(array4)                    # Результат: [1, 3, 5]
8  array5 = array2 + array3
9  array2[2] = 0                # Изменение array2 не повлияло на array5
10 print(array5)                    # Результат: [0, 1, 2, 2, 3, 4]
```

# СПИСКИ СПИСКОВ

Списки кроме стандартных данных типа строк, чисел, также могут содержать другие списки. Подобные списки можно ассоциировать с матрицами и таблицами. Например:

```
1 | array1 = [0, 1, 2]
2 | array2 = [3, 4, 5]
3 | array3 = [6, 7, 8]
4 | array3_3 = [array1, array2, array3]
5 | print(array3_3)          # Результат: [[0, 1, 2], [3, 4, 5], [6, 7, 8]]
6 | print(array3_3[1])       # Результат: [3, 4, 5]
7 | print(array3_3[2][1])    # Результат: 7
```

Чтобы обратиться к элементу вложенного списка, необходимо использовать пару индексов: array3_3[2][1] — обращение ко второму элементу третьего вложенного списка.

$n \times m$

# СПИСКИ СПИСКОВ

Добавление, удаление и изменение общего списка, а также вложенных списков аналогично тому, как это делается с обычными (одномерными) списками. Рассмотрим пример создания матрицы размером n×m, ее заполнения пользователем с клавиатуры и форматированного вывода:

$n \times m$

```
1  n = int(input("Input amount of row"))       # n — количество строк
2  m = int(input("Input amount of column")) # m — количество столбцов
3  matrix=list()         # matrix —матрица размером
4  array = list()        # array — текущая строка матрицы
5  # Заполняем матрицы с клавиатуры
6  for i in range(0, n):
7      for j in range(0, m):
8          elem = float(input(f"Input elem {i};{j}: "))
9          array.append(elem)
10     matrix.append(array.copy())              # кописуем строку в матрицу
11     array.clear()                          # освобождаем строку
12 # Форматированный вывод матрицы на экран
13 for row in matrix:
14     for elem in row:
15         print(elem, end = " ")
16     print(" ")
```

# 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 (для ввода знака присваивания нужно нажать на клавиатуре комбинацию клавиш **<Shift>+<:>** или щелкнуть по кнопке**<:=>** панели **Evaluation**);
2. щелкнуть по кнопке **Vector or Matrix Toolbar** в панели математических инструментов, чтобы открыть панель матричных операций **Matrix**);
3. открыть щелчком по кнопке **Matrix or Vecto**r окно диалога определения размерности матрицы и ввести размерность матрицы: число строк (**Rows)**, число столбцов (**Columns**);
4. закрыть окно диалога, щелкнув по кнопке **Ok**.

- В рабочем документе, справа от знака присваивания, открывается поле ввода матрицы с помеченными позициями для ввода элементов. Для того, чтобы ввести элемент 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;

$x_n$ — ввод нижнего индекса;

$x^{-1}$ — вычисление обратной матрицы;

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

$\overrightarrow{f(M)}$ — поэлементные операции с матрицами:

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

$M^{<>}$ — определение столбца матрицы: $M^{<j>}$ — $j$-й столбец матрицы $M$;

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

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

$\vec{x}\times\vec{y}$ — вычисление векторного произведения векторов:

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

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

$m..n$ — 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.

# 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)*;
- **diag(v)** — создает диагональную матрица, элементы главной диагонали которой хранятся в векторе **v**;
- **identity(n)** — создает единичную матрицу порядка **n**;
- **augment(A, B)** — формирует матрицу, в первых с т о л б ц а х которой содержится матрица **A,** а в последних — матрица **B** (матрицы **A** и **B** имеют одинаковое число строк);
- **stack(A, B)** — формирует матрицу, в первых с т р о к а х которой содержится матрица **A,** а в последних — матрица **B** (матрицы **A** и **B** имеют одинаковое 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**.

# Пример 1. Примеры исполнения функций **matrix, diag, identity, augment, stack, submatrix**

$ORIGIN := 1$

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

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

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

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

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

$E := identity(3) \qquad E = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

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

$F := stack(A,B) \qquad F = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 5 \\ 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 4 \end{pmatrix}$

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

**Функции отыскания различных числовых характеристик матриц:**

- **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**.

# **Пример 2.** Примеры исполнения функций **last, length, rows, cols, max, min, tr, rank, norm1, norm2, norm3**

ORIGIN := 1

$f(x,y) := x + y \qquad A := \text{matrix}(3,4,f) \qquad A = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 5 \end{pmatrix}$

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

$E := \text{identity}(3) \qquad E = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

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

last(v) = 4

длина(v) = 4

rows(D) = 3 cols(D) = 7

max(D) = 5

tr(B) = 10

rank(A) = 2

norm1(B) = 4 norm2(B) = 4

normi(B) = 4 normе(B) = 5.477

**Функции, реализующие численные алгоритмы решения задач линейной алгебры:**

- **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**.

# Пример 3. Примеры исполнения функций **eigenvals, eigenvecs, eigenvec, lsolve**

$$ORIGIN := 1$$

$$C := \begin{pmatrix} 1 & 2 & 3 \\ 0 & 1 & 2 \\ 3 & 4 & 1 \end{pmatrix}$$

$$\text{eigenvals}(C) = \begin{pmatrix} -2.71 \\ 0.271 \\ 5.439 \end{pmatrix}$$

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

$$L := \text{eigenvals}(C)$$

$$\text{eigenvec}(C, L_2) = \begin{pmatrix} -0.76 \\ 0.611 \\ -0.222 \end{pmatrix}$$

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

$n \times m$

# ПРИМЕР РАБОТЫ ПРОГРАММЫ

```
Input amount of row
2
Input amount of column
3
Input elem 0;0:
1
Input elem 0;1:
2
Input elem 0;2:
3
Input elem 1;0:
Input elem 1;1:
5
Input elem 1;2:
6
1.0 2.0 3.0
4.0 5.0 6.0

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

ORIGN := 0

```
Multiply(mas) := | P ← 1
                 | for i ∈ 0 .. length(mas) − 1
                 |     | continue if mas_i = 0
                 |     | P ← P·mas_i
                 | P
```

```
sumPlus(mas) := | sum ← 0
                | for i ∈ 0 .. last(mas)
                |   sum ← sum + mas_i if mas_i > 0
                | sum
```

$$arr := \begin{pmatrix} -7 \\ -5 \\ 4 \\ 1 \\ 0 \\ 1 \\ 3 \\ -2 \\ 4 \\ 0 \end{pmatrix}$$

$arr_0 = -7$ $arr_5 = 1$ $last(arr) = 9$ $length(arr) = 10$

$sumPlus(arr) = 13$ $Multiply(arr) = -3360$

```
firstPlus(mas) := | res ← (-1)
                  |       ( 0)
                  | for i ∈ 0 .. last(mas)
                  |   if mas_i > 0
                  |     | res_0 ← i
                  |     | res_1 ← mas_i
                  |     | break
                  | res
```

$$arr \rightarrow \begin{pmatrix} -7 \\ -5 \\ 4 \\ 1 \\ 0 \\ 1 \\ 3 \\ -2 \\ 4 \\ 0 \end{pmatrix}$$

res := firstPlus(arr)

$$res = \begin{pmatrix} 2 \\ 4 \end{pmatrix}$$

k := $res_0$ $arr_k = 4$

---

vec := ( −7  5  −2  3  1 )

$vec_0 =$ $vec_{0,0} = -7$ $vec_{0,1} = 5$ firstPlus(vec) =

vec := $vec^T$

$$firstPlus(vec) = \begin{pmatrix} 1 \\ 5 \end{pmatrix}$$

```
f(A,B) := | N_A ← rows(A) − 1
          | M_A ← cols(A) − 1
          | N_B ← rows(B) − 1
          | M_B ← cols(B) − 1
          | return −1 if N_A ≠ N_B ∧ M_A ≠ M_B
          | otherwise
          |     res ← 0
          |     for i ∈ 0 .. N_A
          |         for j ∈ 0 .. M_A
          |             res ← res + 1 if A_i,j > B_i,j
          | res
```

$$B := \begin{pmatrix} 3 & 23 & -7 \\ 4 & 24 & -2 \\ 42 & -1 & 19 \\ 12 & 0 & 2 \end{pmatrix} \qquad A := \begin{pmatrix} 1 & 3 & -2 \\ 0 & 4 & 2 \\ 4 & 1 & 9 \\ 5 & 1 & 0 \end{pmatrix}$$

$\text{rows}(A) = 4$ $\qquad f(A,B) = 4$

$\text{cols}(A) = 3$ $\qquad f(B,A) = 8$