Возможности APL

У АПЛ огромное число уникальных св-в и преимуществ. Мы поговорим о основных, т.к. их оооочень много.

Одно из свойств то, что Айверсон, те математические общепринятые операции с обозначениями, оставил так как они есть.

Например:

```
      3+2
5
      3×2
6
      3÷2
1.5
```

Видим, что те математические операци, которые общеприняты, так и остаются общепринятыми.

Чтобы не морочиться со словами, логически False = 0, True = 1

```
      3<2
0
      3>2
1
```

Все общепризнанные знаки сохраняются, например, в теории множеств есть ∈ - принадлежность

Проверим, принадлежит ли 2-ка множеству, которое в правом аргументе.

```
      2∈7 6 2 4
1
```

А 12 - будет нуль.

```
      12∈7 6 2 4
0
```

Т.е. нам не нужно заморачиваться со словами заранее определенными, большинство операций совпадают с общепринятыми в математике.

Теперь важнейшая особенность

```
      3+2 1 5     ⍝ вектор, в котором к каждому эелементу прибавится по тройке
5 4 8
      1 2 3+2 1 5 ⍝ сложение двух векторов
3 3 8
```

Т.е. важное достоинством АПЛ то, что он является ориентированным на работу с массивами любой размерности как с целым.

Нам не надо писать никакие несчастные циклы, мы прямо даем операции в качестве аргументов или два скаляра, или два вектора, или скаляр и вектор и автоматически выполняется или поэлементно эта операция, или, если один из аргументов скаляр, то поэлементно с этим скаляром все выполняется.

Делаем матрицу A и B

```
      A←?3 4⍴9
      B←?3 4⍴9
```

Получили две матрицы 3 на 4

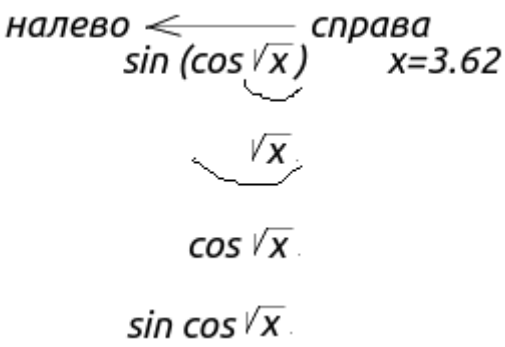
```
      A B
 9 8 2 9  3 3 8 6
 6 9 7 3  8 8 1 6
 5 5 6 8  2 5 1 4
```

```
      A + B ⍝ сложение матриц по элементам, получим одну матрицу 3 на 4
12 11 10 15
14 17  8  9
 7 10  7 12
      A - B ⍝ поэлементное вычисление матриц
 6 5 ¯6  3
¯2 1  6 ¯3
 3 0  5  4
```

Нам плевать, что такое A и B - это два скаляра, два вектора, матрицы, два массива каких-то пятимерных, мы выполним эту операцию без всяких циклов и заморачивания.

Внимание на минус вверху число, потому что это знак числа, чтобы мы его не путали с функцией.

Выполнение в АПЛ идет справа налево. Некоторые чудаки говорят: "Что это такое? Как в Китае".



Первым делом вычислится корень, потом косинус от получившегося числа, а потом уже синус.

Еще один пример на APL:

```
      2+3 4 - 2 1
3 5
```

Проходим 1-ку и 2-ку, никаких ф-ций нет, образовался массив !2 1!. Далее функция "-" (откда вычитать?), идем левее функции, видим 4 и если бы у нас дальше ничего не было, мы бы из 4 отняли 2 и из 4 вычли 1, но мы идем дальше и видим, что это вектор {3;4}. Теперь выполняем первую операцию, от вектора {3;4} вычитаем {2;1}, получится {1;3}. Теперь функция "+" (к чему?), торчит дваечка. левее ничего нет, значит к 2 прибавляем 1 и к 2 прибавляем 3, получаем {3;5}.

Нормальное течение исполнения может быть нарушено круглыми скобками.

Так мы не нарушаем, а облегчаем восприятие:

```
      2+(3 4 - 2 1)
3 5
```

А так нарушаем, получается, что справа вектор {3;4;-2;1} и поэлементно прибавляем 2-ку:

```
      2+3 4 (- 2) 1
3 5
```

Стрелочка влево - знак присваивания, почему не равно? Потому что "=" - это логическая ф-ция, как "<", ">", и т.п.

Иначе:

```
      9 5 1 = 5 ⍝ получаем 9 не равно 5, поэтому 0, 5 равно 5, поэтому 1 и т.д.
0 1 0
```

поэтому присваиваем в X.

```
      ?9 ⍝ черт знает что, от единицы до девяти, ? - генератор случайных чисел
2
      ?9
4
```

```
      x←?9 9 9 9 ⍝ четыре случайных числа, целых, от единицы до девяти
      x
5 1 3 4
```

Как сложить все x? Если у нас тысячи измерений, складывать вручную "+" - не вариант. Для этого потребуется "/" - оператор редукция. *f/x* - это:

$$x_1 f x_2 f \ldots\ldots x_n f$$

Т.е. та функция, которая слева от оператора, помещается в уме между всеми элементами аргумента и после происходит выполнение.

```
      +/x
13
```

Прелесть в том, что с этой редукцией мы можем посылать любую функцию.

```
      ×/x  ⍝ получим вроизведение всех элементов x
60
```

Очень хорошим свойством АПЛ является то, что все переменные носят с собой "набор документов", которые можно спросить и получить информацию о ней.

Например, фунция ρ - размерность, вернет число элементов, если речь идет о векторе

```
      ρx
4
```

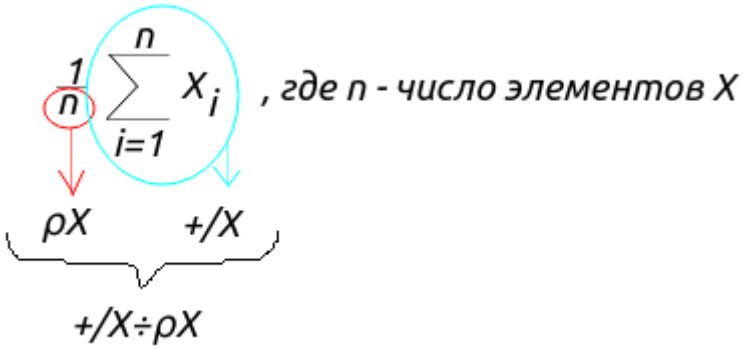
Если будет матрица, то получим вектор {3;4}, первое - число строк, второе - число столбцов

```
      ρA
3 4
```

Чтобы узнать число элементов, то нужно применить " ×/"

```
      ×/ρA
12
```

!!!САМЫЙ ПИСК!!!



Записали среднее как в учебниках математики и как оно будет выглядеть в АПЛ. Получили однуэтажную, очень компактную, математическую функцию.

А дальше СЁКС пошел. Студенты переписали в тетрадь эту строку. Пришли на лабораторные работы и записали ту же строку в АПЛ и нажали Enter. Получили среднее арифметическое вектора x

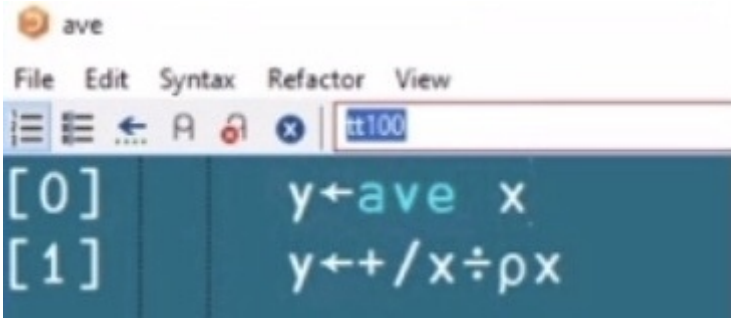
```
      +/x÷ρx
3.25
```

Т.е. АПЛ - это не только сверхкомпактная, математически понятная, без специальных программистских словечек математическая нотация, а это ***ИСПОЛНЯЕМАЯ МАТЕМАТИЧЕСКАЯ НОТАЦИЯ!!!*** Нет границы между записью алгоритма и программой, нет границы между программой и алгоритмом. Это выдающееся и нигде больше неприсутствующееся свойство АПЛ.

Замечание! В АПЛ кроме встроенных или примитивных ф-ций можно определять какие-то свои. В АПЛ нет собственных векторов или значений, если они нам приспичили, можем сделать такую ф-цию сами.

Например:

```
      )ed ave
```



аргументом будет x - вектор, а результатом будет y, и строку со средним присвоим в y.

```
      ave x*2 ⍝ среднее квадратов x
12.75
```

Большинство ф-ций АПЛ могут иметь или 1 аргумент, тогда говорят монадик, например, ρx - одномерное использование ф-ции, или 2 аргумента - доядик (9ρ2 - двуместное использование, сделать вектор длиной 9 из двоек)

```
      ρx
4
      9ρ2
2 2 2 2 2 2 2 2 2
```

Рассмотри как себя ведет подбрасывание монеток

```
      x←?9ρ2 ⍝ 9 случайных чисел от 1-2, т.е. будет вектор из девяти элементов
принимающие значения 1 или 2
1 2 2 2 1 1 1 2 2
```

Герб = 2, тогда в серии из 9 подбрасываний присваиваем 2-ку, получаем 1, где у нас выпала 2, и 0 - где выпала единица

```
      2=x
0 1 1 1 0 0 0 1 1
```

Если проссумируем, то получим сколько раз у нас выпал Герб

```
      +/2=x
5
```

Если разделим на кол-во раз, сколько подбрасывали монетку, сделаем это так

```
      (+/2=?nρ2)÷n←9
0.5555555556
      (+/2=?nρ2)÷n←99
0.5353535354
      (+/2=?nρ2)÷n←99
0.4848484848
      (+/2=?nρ2)÷n←999
0.4734734735
      (+/2=?nρ2)÷n←999
```

```
0.5055055055
      (+/2=?nρ2)÷n←999
0.4834834835
      (+/2=?nρ2)÷n←9999999
0.50019955
      (+/2=?nρ2)÷n←9999999
0.50002765
      (+/2=?nρ2)÷n←9999999
0.50012905
```

присвоили число раз на которое мы подбрасываю а потом поделим на него сумму, то получим ВЕРОЯТНОСТЬ с которой у нас выпадет Герб за эти 9 бросков.

В машинном обучении мы будем использовать АПЛ для демонстрации каких-то алгоритмов обучения, как они и на каких данных работают.

Очень сложной ф-цией является та, которая имеет 5-7 строчек, реализация очень сложного алгоритма, но нам это не грозит

Есть в машинном обучении метод группового учета аргументов, который называют иногда полиминиальными нейронными сетями. На бейсике программа занимала огромное число листов А4 (185см+вытянутая рука и стул и все равно листы лежали на полу), а на АПЛ гватило половинки листа А4.

ML = 3 - мигрейшен левел, системная переменная, для описания системы

IBM ввела обобщенные массивы

```
      ρ(2 3)(2 4 5)(1 2 6 3 2) ⍝ вектор из 3-х элементов, где элементами являются
векторы
3
```

```
      ρ¨(2 3)(2 4 5)(1 2 6 3 2) ⍝ чтобы узнать размерность каждого элемента в
преддущем векторе используем оператор "¨" (ич) - по каждому
2 3 5
```

```
      +/¨(2 3)(2 4 5)(1 2 6 3 2) ⍝ получим сумму элементов элементов вектора
5 11 14
```

```
      ρA B ⍝ вектор длины два
2
      ρ¨A B ⍝ каждый элемент вектора является матрицей 3 на 4
 3 4  3 4
      ]disp A B ⍝ утилитка ]disp графически показывает как устроен обобщенный
массив
┌→──────┬─────────┐
│9 8 2 9│3 3 8 6│
│6 9 7 3│8 8 1 6│
│5 5 6 8↓2 5 1 4↓
└~──────┴~────────┘
```

Обощенные массивы резко повышают возможность реализации любых алгоритмов АПЛ.

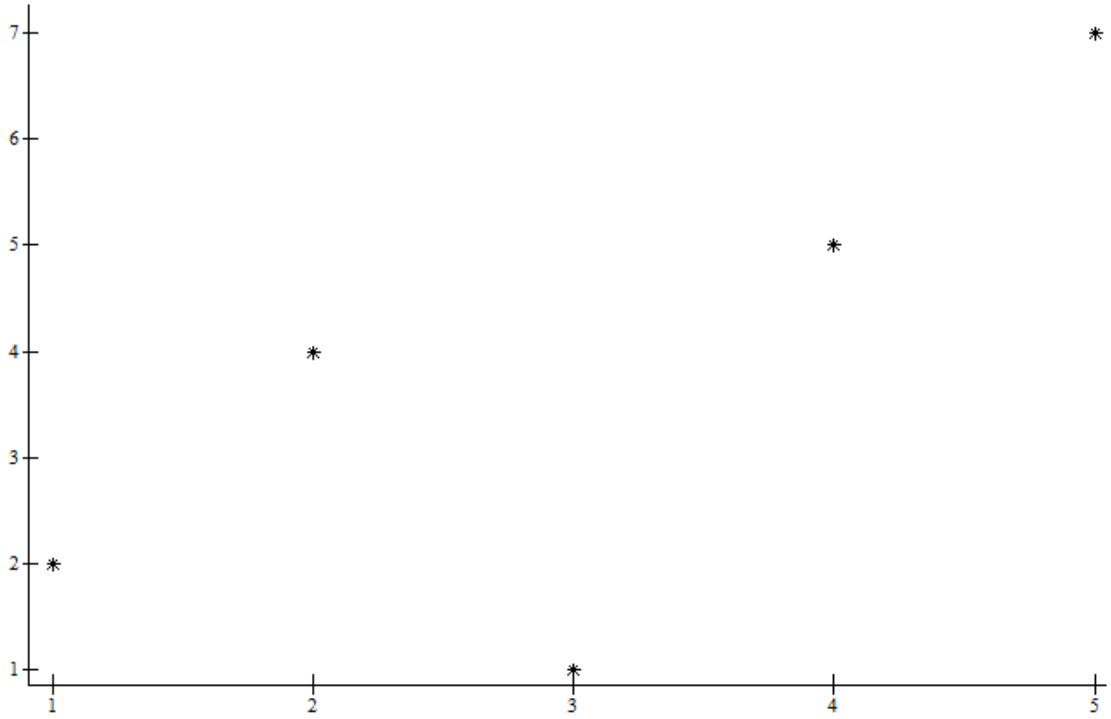
Построение графиков

Подгружаем пакет plt

```
      ]load plt
#.plt
```

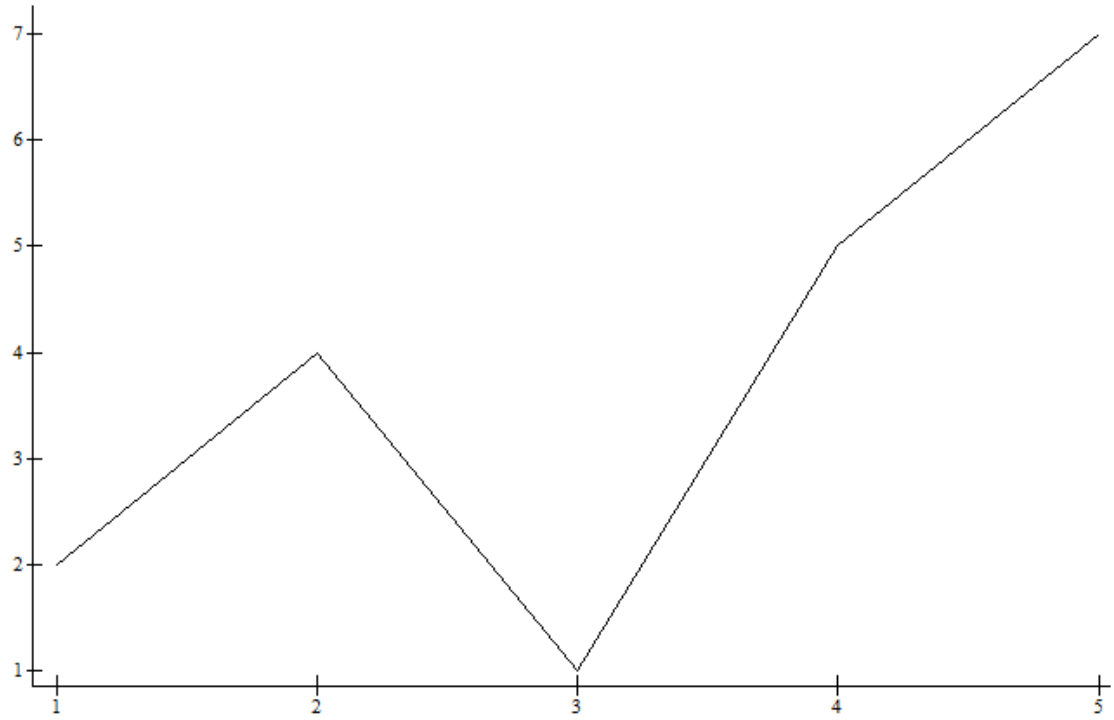
Строим какой-то x точками

```
      x
2 4 1 5 7
      plt.plot x
```



Строим x линиями

```
      1plt.plot x
```



Чтобы не набирать постоянно plt.plot:

```
      ⍝path←'plt' ⍝ path-список мест, где при поиске файла можно не задавать
директорию, где этот файл лежит
```

Можем теперь писать:

```
      plot x
```