
Classes I

Лабораторная работа №1. Списки. Строки и текст.

I.1

1. Используя функцию *Range* создайте список {1,2,3,4}
2. Создайте список из натуральных чисел от 1 до 100
3. Используя функции *Range* и *Reverse*, создайте список {4,3,2,1}
4. Создайте список из натуральных чисел от 1 до 50 в обратном порядке
5. Используя функции *Range*, *Reverse* и *Join*, создайте список {1,2,3,4,4,3,2,1}
6. Нарисовать график списка чисел, которые возрастают от 1 до 100, а затем убывают от 100 до 1
7. Используя функции *Range* и *RandomInteger* создайте список случайно генерированной длины, не превышающей число 10
8. Найти простейшую форму для команды *Reverse[Reverse[Range[10]]]*
9. Найти простейшую форму для команды *Join[{1,2},Join[{3,4},{5}]]*
10. Найти простейшую форму для команды *Join[Range[10],Join[Range[10],Range[5]]]*
11. Найти простейшую форму для команды *Reverse[Join[Range[20],Reverse[Range[20]]]]*

I.2 Strings and Text

Строки и текст (Strings and Text)

В Математике можно вводить текст в виде строки, обозначенной кавычками ("")

"Это строка."

Это строка.

Отметим, что при этом кавычки не отображаются, а отображается только строка. Существует множество функций, которые работают над строками. Как *StringLength*, которая дает длину строки.

StringLength["Утро"]

4

StringReverse меняет символы в строке в обратном порядке

StringReverse["мороз"]

зором

ToUpperCase представляет все символы в строке заглавными буквами

```
ToUpperCase["олимпиада!"]
```

```
ОЛИМПИАДА!
```

StringTake выделяет определенное количество символов, начиная с начала строки

```
StringTake["длинная строка", 10]
```

```
длинная ст
```

StringJoin соединяет строки (не забудьте пробелы, если вы хотите разделить слова)

```
StringJoin["Прекрасная", " ", "погода", " сегодня!"]
```

```
Прекрасная погода сегодня!
```

Выделим первые два символа из каждой строки

```
StringTake[{"груша", "апельсин", "вишня"}, 2]
```

```
{гр, ап, ви}
```

Characters записывает строку в виде списка ее элементов

```
Characters["это простое предложение"]
```

```
{э, т, о, , п, р, о, с, т, о, е, , п, р, е, д, л, о, ж, е, н, и, е}
```

Сортировка символов в строке

```
Sort[Characters["это простое предложение"]]
```

```
{ , , д, е, е, е, е, ж, и, л, н, о, о, о, о, п, п, р, р, с, т, т, э}
```

TextWords дает список слов в строке текста

```
TextWords["это простое предложение"]
```

```
{это, простое, предложение}
```

Список длин каждого слова в строке можно получить следующим образом:

```
StringLength[TextWords["Весна это прекрасное время года."]]
```

```
{5, 3, 10, 5, 4}
```

TextSentences разбивает текстовую строку на список предложений

```
TextSentences["Новое предложение. Оно очень интересное."]
```

```
{Новое предложение., Оно очень интересное.}
```

Получите первые 20 слов из списка общих английских слов

```
Take[WordList[], 20]
```

```
{a, aah, aardvark, aback, abacus, abaft, abalone, abandon, abandoned, abandonment, abase, abasement, abash, abashed, abashment, abate, abatement, abattoir, abbe, abbess}
```

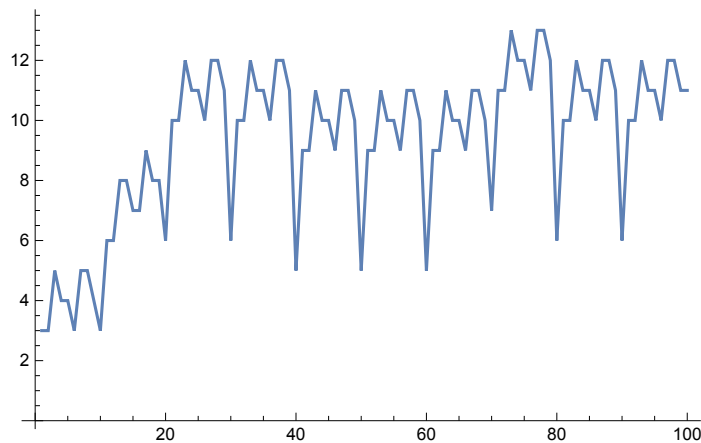
Сгенерируйте строку, дающую название целого числа 56

```
IntegerName[56]
```

```
fifty-six
```

Вот график длин целых чисел от 1 до 100 на английском языке:

```
ListLinePlot[Table[StringLength[IntegerName[n]], {n, 100}]]
```









1. Присоедините две копии строки «Здравствуйте».
2. Создайте единую строку всего английского алфавита заглавными буквами.
3. Сгенерируйте строку английского алфавита в обратном порядке.
4. Объедините 100 копий строки «ABCD»
5. Используйте StringTake, StringJoin и Alphabet, чтобы получить «abcdef»
6. Создайте столбец с числом элементов, равных длине строки «это о строках» и содержащих число букв (включая пробелы) равное номеру элемента.
7. Создайте гистограмму длин слов в «Давным-давно, в далекой галактике, далеко».
8. Найдите максимальную длину слова среди английских слов из WordList []
9. Используйте Length, чтобы найти длину русского алфавита.
10. Создайте строку из заглавных букв греческого алфавита в обратном порядке.

Solutions




Чистые функции (Pure functions)




Начнем с простого примера. Пусть у нас есть список изображений и мы хотим применить к каждому из них функцию Blur:

```
Blur /@ {, , 
```




```
{, , 
```




Теперь предположим, что мы хотим включить параметр 5 в Blur. Как мы можем сделать это? Ответ заключается в использовании чистой функции.

`Blur[#, 5] & /@ {`  `,`  `,`  `}`




`{`  `,`  `,`  `}`




Первоначальное размытие (blur), записывается как чистая функция:

`Blur[#] & /@ {`  `,`  `,`  `}`

`{`  `,`  `,`  `}`

Это эквивалент `Blur[#, 5]& /@`




`{Blur[`  `, 5], Blur[`  `, 5], Blur[`  `, 5]}`

`{`  `,`  `,`  `}`

Каждый раз слот (#) указывает, куда поместить каждый элемент, когда применяется чистая функция.





Повернем каждую строку на 90 °:

`Rotate[#, 90 Degree] & /@ {"one", "two", "three"}`

`{`  `,`  `,`  `}`

Возьмем строку и повернем ее на различные углы:

`Rotate["рассвет", #] & /@ {30 °, 90 °, 180 °, 220 °}`

`{`  `,`  `,`  `,`  `}`

Изобразим текст в списке окрашенным в разные цвета:

`Style["предложение", 20, #] & /@ {Red, Orange, Blue, Purple}`

`{`  `,`  `,`  `,`  `}`






Построим таблицу цветов с оттенками от 0 до 1:

`Table[Hue[x], {x, 0, 1, 0.05}]`

`{`  `,`  `,`  `,`  `,`  `,`  `,`  `,`  `,`  `,`  `,`  `,`  `}`

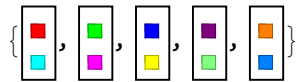
Нарисуем окружности различных размеров

`Graphics[Circle[], ImageSize -> #] & /@ {20, 40, 30, 50, 10}`

`{`  `,`  `,`  `,`  `,`  `}`

Изобразим столбцы в рамке, содержащие цвета и их отрицания:

```
Framed[Column[{#, ColorNegate[#]}]] & /@ {Red, Green, Blue, Purple, Orange}
```



Образуем список цифр, а затем применим к нему чистую функцию:

```
Style[#, Hue[#/10], 5 * #] & /@ IntegerDigits[2^100]
```

```
{1, 2, 6, 7, 6, 5, , 6, , , 2, 2, 8, 2,
 2, 9, 4, , 4, 9, 6, 7, , 3, 2, , 5, 3, 7, 6}
```

Покажем действие чистой функции на список {6, 8, 9}:

```
{Style[6, Hue[6/10], 5 * 6], Style[8, Hue[8/10], 5 * 8], Style[9, Hue[9/10], 5 * 9]}
```

```
{6, 8, 9}
```

Приведем теперь более абстрактные примеры.

Действие абстрактной чистой функции на список:

```
f[#, x] & /@ {a, b, c, d, e}
```

```
{f[a, x], f[b, x], f[c, x], f[d, x], f[e, x]}
```

более простой пример:

```
f[#] & /@ {a, b, c, d, e}
```

```
{f[a], f[b], f[c], f[d], f[e]}
```

эквивалентная запись

```
f[#] & {a, b, c, d, e}
```

```
f[a]
```

Применим чистую функцию к x, так что # slot заменяется на x:

```
f[#, a] & [x]
```

```
f[x, a]
```

эквивалентная форма записи

```
f[#, a] &@x
```

```
f[x, a]
```

Итак, теперь мы можем видеть, что делает / @: он просто применяет чистую функцию к каждому элементу в списке.

```
f[#, a] & /@ {x, y, z}
```

```
{f[x, a], f[y, a], f[z, a]}
```

эквивалентная запись

```
{f[#, a] &@x, f[#, a] &@y, f[#, a] &@z}
```

```
{f[x, a], f[y, a], f[z, a]}
```

I.3

1. Используйте Range и чистую функцию для создания списка из квадратов первых 20 натуральных чисел.
2. Составьте список из результатов смешивания желтого, зеленого и синего цветов с красным цветом.
3. Создайте список столбцов в рамках, содержащих прописные и строчные версии каждой буквы алфавита.
4. Составьте список букв алфавита в случайных цветах в рамках, имеющих случайные цвета фона.
5. Напишите в более простой форме команду $(\#^2+1\&)/@Range[10]$.

Solutions

I.4

Композиции функций и их программирование

Выражение $f[x]$ определяет применение функции f к переменной x . $f[f[x]]$ определяет применение функции f к $f[x]$ или определяет композицию $f \circ f$.







Следующий список содержит переменную x , функцию $f[x]$ и результаты последовательного действия этой функции до 4 раз.

```
NestList[f, x, 4]
```

```
{x, f[x], f[f[x]], f[f[f[x]]], f[f[f[f[x]]]]}
```

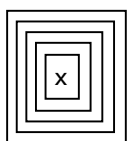
Использование опции Framed как функции делает его более наглядным; добавим функцию NestList:

```
NestList[Framed, x, 5]
```

```
{x, , , , , , }
```

Результат действия 5-ти раз функции имеет вид:

```
Nest[Framed, x, 5]
```



Применяя EdgeDetect к изображениям, сначала находим края, затем края ребер и так далее.

```
NestList[EdgeDetect, , 6]
```



Начнем с красного цвета и смешивая его с желтым на каждой итерации, будем получать все более желтый цвет.

```
NestList[Blend[{#, Yellow}] &, Red, 20]
```



Если последовательно применять функцию, которая добавляет к предыдущему числу 1, то в результате получаем только целые числа.

```
NestList[#+1 &, 1, 15]
```

```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16}
```

Последовательно умножая на 2, получим степени 2. Список из 1 и первых 15-ти степеней двойки имеет вид

```
NestList[2 * # &, 1, 15]
```

```
{1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768}
```

Последовательно применим функцию квадратного корня

```
NestList[Sqrt[1 + #] &, 1, 5]
```

$$\{1, \sqrt{2}, \sqrt{1+\sqrt{2}}, \sqrt{1+\sqrt{1+\sqrt{2}}}, \sqrt{1+\sqrt{1+\sqrt{1+\sqrt{2}}}}, \sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{2}}}}}\}$$

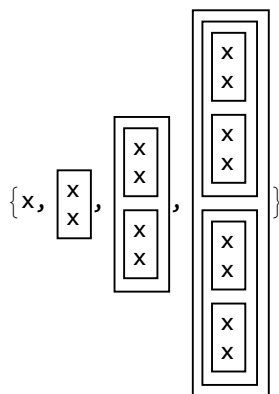
Представление результата в десятичной форме быстро сходится (к величине золотого сечения):

```
NestList[Sqrt[1 + #] &, 1, 10] // N
```

```
{1., 1.41421, 1.55377, 1.59805, 1.61185,
 1.61612, 1.61744, 1.61785, 1.61798, 1.61802, 1.61803}
```

Вложенные ящики рекурсивно объединяются по два на каждом уровне:

```
NestList[Framed[Column[{#, #}]] &, x, 3]
```

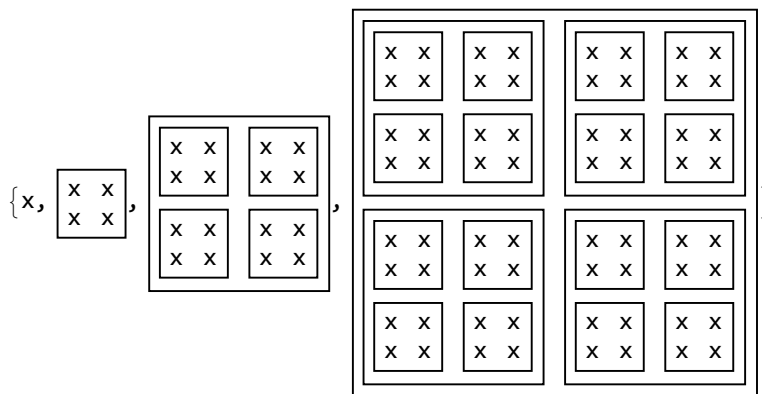


Определим список, элементы которого равны числу букв в словах предложения:

```
StringLength[TextWords["Весна это прекрасное время года."]]
{5, 3, 10, 5, 4}
```

TextSentences разбивает текстовую строку на список предложений

```
NestList[Framed[Grid[{{#, #}, {#, #}}]] &, x, 3]
```



Добавим 0 в список, а затем добавим суммы соседних элементов:

```
NestList[Join[{0}, #] + Join[#, {0}] &, {1}, 5]
{{1}, {1, 1}, {1, 2, 1}, {1, 3, 3, 1}, {1, 4, 6, 4, 1}, {1, 5, 10, 10, 5, 1}}
```

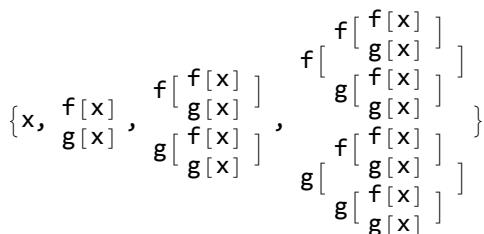
Если поместить этот результат в сетку, то он образует треугольник Паскаля с биномиальными коэффициентами:

```
NestList[Join[{0}, #] + Join[#, {0}] &, {1}, 8] // Grid
```

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
```

Упорядочим рекурсивную структуру в столбцах:

```
NestList[Column[{f[#], g[#]}] &, x, 3]
```



1. Составьте список результатов использования функции Blur до 10 раз, начиная с растрованного размера 30 "X" (Используйте: Rasterize[Style]).
2. Начните с x, затем создайте список, использовав последовательно функцию Framed до 10 раз, используя каждый раз случайный цвет фона.
3. Начните с размера 50 для "A", затем создайте список вложенного применения Framed и

случайного вращения 5 раз.

4. Создайте линейный график из 100 итераций логического итерационного отображения $4 \#(1 - \#)&$, начиная с 0.2.
5. Найдите числовое значение результата из 30 итераций применения функции $1 + 1/\#&$, начиная с 1
6. Создайте список из первых 10 степеней 3 (начиная с 0) путем вложенного умножения.
7. Составьте список результатов действия вложенной функции (метод Ньютона) $(\# + 2 / \#) / 2$ до 5 раз, начиная с 1.0, а затем вычтите $\sqrt{2}$ из всех результатов.
8. Создайте список из 5 степеней числа 2, т. е. $2^0, 2^1, 2^2, \dots, 2^n$ раз, причем n меняется от 0 до 4.

I.5

Определение функций пользователя

Существует большое число функций, которые уже встроены в язык Математики. Но имеются также широкие возможности для того, чтобы определить свои собственные функции. И язык Wolfram Language очень удобен для этого.

Определим функцию `pinks`, которая принимает любой аргумент:

```
pinks[n_] := Table[Pink, n]
```

Используем введенное определение функции:

```
pinks[5]
```

```
{Pink, Pink, Pink, Pink, Pink}
```

```
pinks[10]
```

```
{Pink, Pink, Pink, Pink, Pink, Pink, Pink, Pink, Pink, Pink}
```

Как работает это определение функции? Идея состоит в том, что знак `:=` определяет значение для розовых квадратов `[n_]`. Когда мы вводим `pinks[5]`, это соответствует шаблону `pinks[n_]`. Затем используется значение, которое мы определили для этого.

Вот список выражений:

```
{f[Red], f[Yellow], f[Green], f[Orange], f[Magenta]}
```

```
{f[Red], f[Yellow], f[Green], f[Orange], f[Magenta]}
```

Определим значения `f[Red]` и `f[Green]`:

```
f[Red] = 1000; f[Green] = 2000;
```

Теперь `f[Red]` и `f[Green]` заменяются соответствующими числовыми значениями; остальные элементы списка остаются неизменными:

```
{f[Red], f[Yellow], f[Green], f[Orange], f[Blue]}
```




```
{1000, f[Yellow], 2000, f[Orange], f[Blue]}
```

Теперь добавим определение для шаблона `f[x_]`.

```
f[x_] := Framed[Column[{x, ColorNegate[x]}]]
```

Если специальные случаи не рассматриваются (с определенными значениями аргумента), то используется общее определение:

```
{f[Red], f[Yellow], f[Green], f[Orange], f[Blue]}
```

```
{1000, , 2000, , 
```

Очистим введенные определения для `f`, чтобы позже можно было использовать это обозначение

```
Clear[f]
```

Второй пример: определим значения для функции факториала. Начните с того, что `factorial[1] = 1`. Затем определим, как вычислить `factorial[n_]` рекурсивно в терминах факториала с меньшим значением аргумента.

```
factorial[1] = 1; factorial[n_Integer] := n * factorial[n - 1]
```

Найдем значение `factorial[50]`:

```
factorial[50]
```

```
30 414 093 201 713 378 043 612 608 166 064 768 844 377 641 568 960 512 000 000 000 000
```

Имеется также встроенная факториальная функция, которая дает тот же результат:

```
50!
```

```
30 414 093 201 713 378 043 612 608 166 064 768 844 377 641 568 960 512 000 000 000 000
```

Вместо определения для `factorial[1]` и `factorial[n_]` мы могли бы ввести одно определение и использовать `If`. Но наличие отдельных определений для каждого случая позволяет упрощать чтение и понимание действия функции.

Альтернативное определение функции с использованием условия `If`:

```
factorial[n_Integer] := If[n == 1, 1, n * factorial[n - 1]]
```

Рассмотрим простой пример создание определения для `plusminus[{x_, y_}]`. Определим значение для шаблона

```
plusminus[{x_, y_}] := {x + y, x - y}
```

Воспользуемся введенным определением

```
plusminus[{4, 1}]
```

```
{5, 3}
```

Менее элегантная форма, основана на традиционном определении функции `function[argument]`:

```
In[1]:= plusminus[v_] := {v[[1]] + v[[2]], v[[1]] - v[[2]]}
```

Определим функцию, которая применяется только к обведенным в рамку объектам:

```
In[2]:= highlight[Framed[x_]] := Style[Labeled[x, "+"], 20, Background -> LightYellow]
```

Применить выделение для каждого элемента списка;

```
In[3]:= highlight /@ {a, Framed[b], c, Framed[{10, 20}], 100}
```

```
Out[3]= {highlight[a], b, highlight[c], {10, 20}, highlight[100]}
      +
      +
```

Это определение относится ко всему списку:

```
highlight[list_List] := highlight /@ list
```

Теперь больше не нужно использовать символы / @:

```
highlight[{a, Framed[b], c, Framed[{10, 20}], 100}]
```

```
{highlight[a], b, highlight[c], {10, 20}, highlight[100]}
      +
      +
```

Дадим общее определение рассматриваемой функции, чтобы показать, что ни один из ее специальных случаев не требует отдельного рассмотрения:

```
highlight[x_] := Style[Rotate[x, -30 Degree], 20, Orange]
```

Для возникающих специальных случаев сразу же вычисляются значения функции:

```
highlight[{a, Framed[b], c, Framed[{10, 20}], 100}]
```

```
{a, b, c, {10, 20}, 100}
      +
      +
```

1. Определите функцию f , которая вычисляет квадрат ее аргумента.
2. Определите функцию $poly$, которая имеет целочисленное значение аргумента и создает изображение оранжевого правильного многоугольника с числом сторон, равным значению этого аргумента.
3. Определите функцию f , которая берет список из двух элементов и помещает их в обратном порядке.
4. Определите функцию f , которая имеет два аргумента и дает результат в виде дроби, где числитель равен произведению этих элементов, а знаменатель - их сумме.
5. Определите функцию f , которая имеет два аргумента и дает результат в виде списка трех элементов, которые соответственно равны сумме, разности и частному этих аргументов.
6. Определите функцию $evenodd$, которая зависит от одного целочисленного аргумента и имеет значение `Black`, если ее аргумент четный, значение - `White` в противном случае, значение `Red` - если аргумент равен нулю.
7. Определите функцию $Fibonacci$ с $f[0]=1$ и $f[1]=1$, а $f[n]$ для натурального числа n является суммой $f[n-1]$ и $f[n-2]$.

Solutions