

Разработайте на JavaScript программу «Калькулятор», позволяющую объявлять переменные и функции и выполнять арифметические операции над ними. Программа может быть разработана как для Node.JS (консольное приложение), так и для браузера (одностраничное приложение с графическим интерфейсом, в простейшем случае интерфейс может состоять из текстового поля ввода, кнопки «Добавить» и области, в которую выводится результат работы программы).

## Требования к коду

Решение должно использовать объектно-ориентированный подход, содержать подходящие классы и структуры, моделирующие сущности предметной области. Примените подходящие для решения задачи контейнеры, встроенные в JavaScript.

Наличие автотестов для программы в целом или для её классов и функций приветствуется. Вы можете использовать привычный вам фреймворк для написания тестов.

Используйте привычный вам стиль именования классов, переменных, методов и функций.

## Формат входных и выходных данных

Входные данные поступают из построчно (в консольном приложении из `stdin`, в браузерном приложении из поля ввода графического интерфейса). Выходные данные выводятся: в `stdout` в консольном приложении либо в область результатов в графическом интерфейсе одностраничного приложения (например, в текстовое поле только для чтения). Каждая входная строка содержит одну из показанных ниже команд. Каждая команда выполняется сразу после считывания.

Пример интерфейса программы в случае разработки браузерного приложения:

Page 1

Здесь вводятся команды

Command

let x=3

Submit

Input:

```
var x
var y
fn xPlusY=x+y
print xPlusY
```

Output:

nan

Здесь отображаются введенные команды

Здесь отображаются выводимые данные

Список команд:

- **var <идентификатор>**  
**Объявляет переменную** типа double с именем <идентификатор>. Идентификатор **не должен совпадать** ни с одним из **ранее объявленных** имен переменных и функций. Значение переменной после ее объявления не определено (можно использовать значение NAN (not a number) для этих целей). В отсутствие ошибок в stdout ничего не выводится. При наличии ошибки команда игнорируется, а в stdout выводится сообщение об ошибке.
- **let <идентификатор1>=<число с плавающей запятой>** либо **let <идентификатор1>=<идентификатор2>**  
Присваивает **переменной** с именем <идентификатор1> числовое значение, либо **текущее значение ранее объявленного** идентификатора с именем <идентификатор2>. Если переменная с именем <идентификатор1> не была ранее объявлена, происходит

объявление новой переменной. В качестве *<идентификатора1>* не может выступать имя функции. В отсутствие ошибок в stdout ничего не выводится. В случае ошибки команда игнорируется, а в stdout выводится сообщение об ошибке.

- **fn *<идентификатор1>*=*<идентификатор2>* либо *<идентификатор1>*=*<идентификатор2>**<операция>**<идентификатор3>***  
Объявляет новую функцию с ранее **необъявленным именем** *<идентификатор1>*, значением которой будет либо значение идентификатора *<идентификатор2>*, либо результат применения одной из следующих бинарных *операций* к значениям **ранее объявленных** идентификаторов *<идентификатор2>* и *<идентификатор3>* **в момент вычисления значения функции**:
  - **+** Сложение. Результат – сумма значений *<идентификатор2>* и *<идентификатор3>*.
  - **-** Вычитание. Результат – разность значений *<идентификатор2>* и *<идентификатор3>*.
  - **\*** Умножение. Результат – произведение значений *<идентификатор2>* и *<идентификатор3>*.
  - **/** Деление. Результат – частное значений *<идентификатор2>* и *<идентификатор3>*.
  - Если значение хотя бы одного из операндов операции не определено, результатом операции должно быть неопределенное значение (NaN). В отсутствие ошибок в stdout ничего не выводится. В случае ошибки команда игнорируется, а в stdout выводится сообщение об ошибке.
- **print *<идентификатор>***  
Выводит в stdout значение **ранее объявленного** идентификатора. Если идентификатором являлась переменная, то выводится ее значение, а если функция, то выводится вычисленное значение функции. Значение идентификатора **выводится** с точностью в 2 знака после запятой. Например, число 0.33333 должно быть выведено как 0.33. Если значение идентификатора не определено, должно быть выведено **nan**. В случае ошибки (например, попытка вывести значение необъявленного идентификатора), команда игнорируется, а в stdout должен быть выведено сообщение об ошибке.
- **printvars**  
Выводит в stdout имена и значения всех ранее объявленных переменных, **отсортированных по алфавиту**, по одному в каждой строке в следующем формате:  
*<идентификатор>*:*<значение>*  
Значение переменной выводится с точностью **2 знака после запятой**. Если значение переменной не определено, должно быть выведено **nan**. Если ни одной переменной не было объявлено к моменту выполнения команды printvars, в stdout выводиться ничего не должно.
- **printfns**  
Выводит в stdout имена и значения всех ранее объявленных функций, **отсортированных по алфавиту**, по одному в каждой строке в следующем формате:  
*<идентификатор>*:*<значение>*  
Значение функции выводится с точностью **в 2 знака после запятой**. Если значение функции не определено, должно быть выведено **nan**. Если ни одной функции не было объявлено к моменту выполнения команды printfns, в stdout выводиться ничего не должно

**Идентификатор** – непустая строка, в которой можно использовать буквы английского алфавита, цифры и символ подчеркивания. Идентификатор не может начинаться с цифры. Идентификаторы используются в качестве имен переменных и функций.

## Примеры

### Объявление, присваивание и вывод значений переменных

Ввод	Вывод	Пояснение
var x		Объявляем переменную <b>x</b>
print x	nan	Значение переменной <b>x</b> пока не определено
let x=42		Присваиваем переменной <b>x</b> значение 42
print x	42.00	Теперь <b>x</b> хранит значение 42
let x=1.234		Значение переменной можно изменить
print x	1.23	Значение выводится с точностью 2 знака после запятой
let y=x		Автоматически объявляем переменную <b>y</b> и присваиваем ей текущее значение <b>x</b>
let x=99		Присваиваем переменной <b>x</b> значение 99
printvars	x:99.00 y:1.23	Переменная <b>y</b> хранит присвоенное ей значение <b>x</b> . Последующие манипуляции над <b>x</b> не оказывают на нее влияния. Переменные выводятся в алфавитном порядке

### Объявление функций

Ввод	Вывод	Пояснение
var x		
var y		
fn XPlusY=x+y		
print XPlusY	nan	Значение функции не определено, т.к. не определены значения ее аргументов
let x=3		
let y=4		

print XPlusY	7.00	Теперь значение функции определено
let x=10		
print XPlusY	14.00	Значение функции зависит от значений ее аргументов
let z=3.5		
fn XPlusYDivZ=XPlusY/z		Значение функции может зависеть не только от значений переменных, но и от значений других функций
printfns	XPlusY:14.00 XPlusYDivZ:4.00	Значения функций выводятся в алфавитном порядке

### Различия между fn и let

Ввод	Вывод	Пояснение
let v=42		
let variable=v		variable хранит значение v (42)
fn function=v		function хранит действие, которое будет вычислено при получении значения функции
let v=43		
print variable	42.00	Переменная variable хранит ранее присвоенное значение 42
print function	43.00	Значением function будет значение переменной v, вычисленное в момент вызова функции (а не ее объявления)

### Пример: вычисление площади круга

Ввод	Вывод	Пояснение
var radius		
let pi=3.14159265		
fn radiusSquared=radius*radius		
fn circleArea=pi*radiusSquared		

let radius=10		
print circleArea	314.16	
let circle10Area=circleArea		circle10Area хранит значение функции circleArea, вычисленной при radius=10
let radius=20		
let circle20Area=circleArea		circle20Area хранит значение функции circleArea, вычисленной при radius=20
printfns	circleArea:1256.64 radiusSquared:400.00	
printvars	circle10Area:314.16 circle20Area:1256.64 pi:3.14 radius:20.00	

### Пример: вычисление последовательности Фибоначчи

Ввод	Вывод	Пояснение
let v0=0		
let v1=1		
fn fib0=v0		
fn fib1=v1		
fn fib2=fib1+fib0		
fn fib3=fib2+fib1		
fn fib4=fib3+fib2		
fn fib5=fib4+fib3		
fn fib6=fib5+fib4		
printfns	fib0:0.00 fib1:1.00 fib2:1.00 fib3:2.00 fib4:3.00 fib5:5.00	

	fib6:8.00	
let v0=1		
let v1=1		
printfns	fib0:1.00 fib1:1.00 fib2:2.00 fib3:3.00 fib4:5.00 fib5:8.00 fib6:13.00	

## Дополнительное задание: оптимизация

Наивный подход к вычислению значений функций в некоторых ситуациях может приводить длительной работе программы. Например, вычисление значения хотя бы 50-го числа последовательности Фибоначчи с использованием функций **fib0**, **fib1**, **fib2**, ..., **fib50** может занимать несколько секунд, или даже минут. Если ваше решение страдает от этой проблемы, усовершенствуйте программу, устранив причину долгой работы.