GeekBrains

Основы программирования

Циклы

[В какой версии ПО был написан код. Если актуально]

Иллюстрации для обложки:

https://www.dropbox.com/sh/lzla84g77kbk851/AAASYvRaXE-4idVbxkXaC4LYa/Illustration?dl=0&subfolder_nav_tracking=1

Иллюстрация должна быть отцентрирована, уменьшать в зависимости от размера заголовка

Акцентный для выделений: #6654D9

На этом уроке

1. Узнаем что такое циклические алгоритмы и какие задачи они решают;
2. Узнаем как можно визуализировать циклические алгоритмы с помощью блок-схем;
3. Научимся использовать циклы в программах;
4. Научимся генерировать случайные числа;
5. Узнаем об области видимости переменных.

Оглавление

Введение

Игра «Угадай число»

- Случайные числа
- Реализация алгоритма

Цикл с постусловием

Область видимости переменных

Задача нахождения факториала

Применение цикла for

Шахматная доска

Домашнее задание

- Дополнительно

Глоссарий

Дополнительные материалы

Используемые источники

Введение

На прошлом уроке вы узнали, что такое ветвление и научились применять его на практике, что позволило решать задачи, связанные с проверкой различных условий. Чтобы научиться решать более сложные задачи, нам нужно научиться создавать циклические алгоритмы. В программировании постоянно возникают задачи, требующие выполнения одного и того же или похожего действия много раз. Например, для того, чтобы поздравить всех покупателей в интернет-магазине, нужно для каждого в шаблонный текст подставить имя и инициировать отправку письма на email пользователя. Для каждого пользователя действие повторяется в точности. Именно для выполнения повторяющихся действий и нужны циклы.

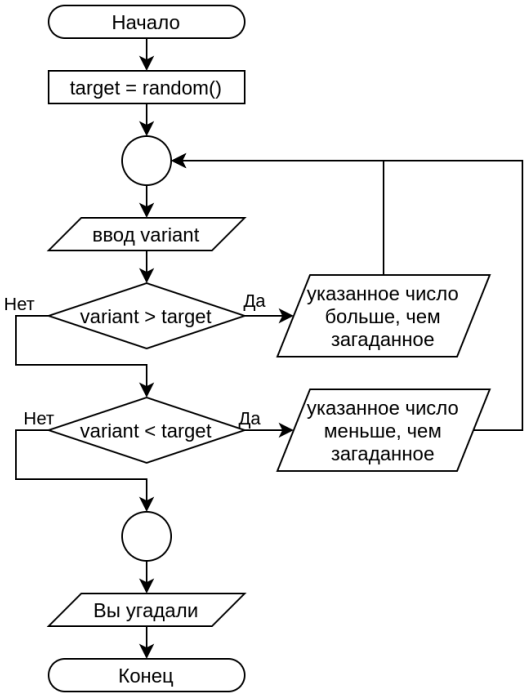
Игра «Угадай число»

Один из самых удачных примеров задачи, при решении которой не обойтись без цикла — это игра «Угадай число». Её условия очень просты: компьютер загадывает какое-то случайное число, а задача пользователя его отгадать. Каждый раз, когда пользователь вводит новый вариант, программа выводит одно из следующих сообщений:

- Указанное число больше, чем загаданное;
- Указанное число меньше, чем загаданное;
- Вы угадали!

Выполнение программы завершается только в случае угадывания числа. В других случаях она снова и снова запрашивает у пользователя вариант ответа — это и есть цикл.

Составим блок-схему алгоритма, решающего данную задачу:



На этой блок-схеме описано следующее поведение: вначале программы генерируется случайное число, его значение записывается в переменную target, после чего алгоритм входит в цикл. В цикле от пользователя требуется ввести вариант ответа, введённое значение помещается в переменную variant. Если variant больше target, выводится соответствующее сообщение и происходит возврат в начало тела цикла. В противном случае, если variant меньше target, то выводится соответствующее сообщение и происходит возврат в начало тела цикла. В противном случае выводится сообщение о том, что число угадано и выполнение программы завершается.

Для того, чтобы написать код, реализующий эту логику, нам нужно разобраться с тем, как генерировать случайные числа.

Случайные числа

Для генерации случайных чисел в JavaScript используется функция Math.random(), она проста в использовании:

```
console.log(Math.random()); // 0.47061591587212837
console.log(Math.random()); // 0.7163555254447114
console.log(Math.random()); // 0.29164797332541825
```

Как видно, эта функция каждый раз возвращает случайное число в диапазоне от 0 включительно до 1 невключительно. Если выполнить этот же код ещё раз, результаты будут совсем другими:

```
console.log(Math.random()); // 0.6704727238059762
console.log(Math.random()); // 0.36088913118803534
```

```
console.log(Math.random()); // 0.7830182610573135
```

Поскольку для нашей задачи нам нужны числа больше, например 5, 9, то полученный с помощью Math.random() результат необходимо умножить на верхний порог требуемого нам диапазона, например, на 10:

```
console.log(Math.random() * 10); // 6.503905930411702
console.log(Math.random() * 10); // 8.240538361999661
console.log(Math.random() * 10); // 2.779736759325453
```

Остаётся округлить числа, чтобы они стали целыми. Сделать это можно с помощью функции Math.round, которая округляет число до целого:

```
console.log(Math.round(Math.random() * 10)); // 4
console.log(Math.round(Math.random() * 10)); // 10
console.log(Math.round(Math.random() * 10)); // 5
```

Реализация алгоритма

Теперь, когда мы можем генерировать случайные целые числа в нужном нам диапазоне, можно приступить к написанию кода, реализующего логику алгоритма, представленного на блок-схеме:

```
let target = Math.round(Math.random() * 10);

while (true) {
  let variant = prompt('Укажите вариант ответа');

  if (variant > target) {
    console.log('Число ' + variant + ' больше, чем загаданное');
  } else if (variant < target) {
    console.log('Число ' + variant + ' меньше, чем загаданное');
  } else {
    break;
  }
}

console.log('Вы угадали число ' + target + '!');
```

В коде появился вызов оператора **while**, который очень похож на вызов оператора **if**, который мы изучали на прошлом уроке. Действительно, синтаксис у него похож: в скобках необходимо указывать условие. Цикл будет выполняться снова и снова до тех пор, пока условие в скобках **while** будет истинным. Можно было бы даже написать так, и программа работала бы точно так же:

```
while (1 > 0) {
```

Потому что, как мы выяснили в прошлом уроке, операторы сравнения возвращают булево значение, которое может быть true или false:

```
console.log(1 > 0) // true
console.log(1 < 0) // false
```

Поскольку условие **true** оператора **while** всегда выполняется, цикл считается бесконечным и его выполнение может быть прервано только вызовом **break**, что и происходит в блоке **else**.

Точно так же как и у оператора **if**, у блока **while** тоже есть тело — код, который выделяется фигурными скобками **{ }**. Именно этот код и выполняется циклично. Каждое новое выполнение кода цикла называется **итерацией**.

Цикл с постусловием

У оператора **while** есть напарник, оператор **do while**. Перепишем код на использование **do while** вместо **while**:

```
let target = Math.round(Math.random() * 10);
let variant;

do {
  variant = prompt('Укажите вариант ответа');

  if (variant > target) {
    console.log('Число ' + variant + ' больше, чем загаданное');
  } else if (variant < target) {
    console.log('Число ' + variant + ' меньше, чем загаданное');
  }
} while (variant != target);

console.log('Вы угадали число ' + target + '!');
```

Разница между этими операторами в том, что тело оператора **do while** выполняется как минимум один раз, независимо от условия. В предыдущем примере нам приходилось писать условие while (true), чтобы добиться такого поведения. Кроме того, **do while** проверяет условие в конце. То есть, **while** проверяет условие перед итерацией, а **do while** в конце, поэтому их так и называют:

- **while** — цикл с предусловием;
- **do while** — цикл с постусловием.

В конце каждой итерации будет проверено условие variant != target. Если variant будет равен 5, а target 6, то проверка 5 != 6 вернёт значение true (утверждение, что пять не равно шести является правдой) и, так как условие выполняется, будет произведена ещё одна итерация. А вот когда значения совпадут, например 6 != 6 (утверждение, что шесть не равно шести является ложью), это вернёт false и цикл будет завершён.

Засчёт перестроения логики с использования **while** на использование **do while**, мы даже избавились от **else**, потому что **break** нам больше не нужен. С другой стороны, объявление переменной variant пришлось вынести за пределы цикла и вот почему.

Область видимости переменных

Доступность переменной зависит от того, где она объявлена. Если она объявлена внутри какого-то блока, то снаружи или в соседнем блоке она просто не будет видна:

```
if (true) {
  let x = 0;
}
console.log(x); // ОШИБКА: x is not defined
```

Поэтому переменную нужно объявлять на том уровне, на котором она будет использоваться. Кроме того, она будет доступна и во вложенных блоках:

```
let x = 0;
if (true) {
  console.log(x); // 0
  x = 1;
}
console.log(x); // 1
```

Если бы мы не вынесли переменную variant на уровень, где объявлен цикл, то получили бы ошибку, поскольку использовали бы её в условии **while**, хотя объявлена она в блоке **do { ... }**:

```
let target = Math.round(Math.random() * 10);

do {
  let variant = prompt('Укажите вариант ответа');

  if (variant > target) {
    console.log('Число ' + variant + ' больше, чем загаданное');
  } else if (variant < target) {
    console.log('Число ' + variant + ' меньше, чем загаданное');
  }
} while (variant != target); // ОШИБКА: variant is not defined
```

```
console.log('Вы угадали число ' + target + '!');
```

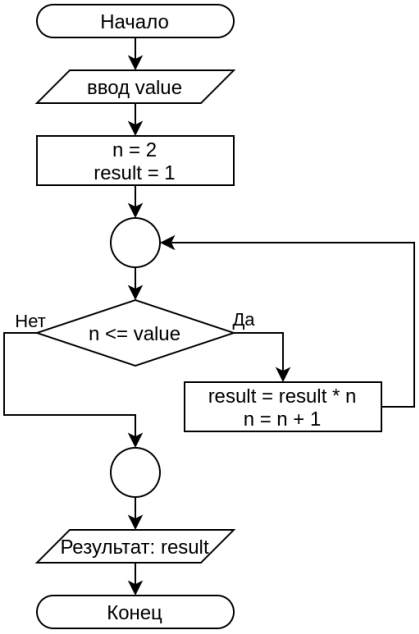
Впрочем, мы могли бы продолжать использовать уже знакомый нам приём с while (true), и вернуться к применению **break**:

```
let target = Math.round(Math.random() * 10);

do {
  let variant = prompt('Укажите вариант ответа');

  if (variant > target) {
    console.log('Число ' + variant + ' больше, чем загаданное');
  } else if (variant < target) {
    console.log('Число ' + variant + ' меньше, чем загаданное');
  } else {
    break;
  }
} while (true);

console.log('Вы угадали число ' + target + '!');
```

В программировании одну и ту же задачу можно решить множеством способов. Программирование даёт большой простор для творчества, этим оно и прекрасно.

Задача нахождения факториала

Решим задачу нахождения факториала числа. Факториал числа n определяется как произведение всех натуральных чисел от 1 до n включительно, то есть факториал 5 это 1 * 2 * 3 * 4 * 5 = 120. Так же, нужно помнить, что факториал 0 = 1. Составим блок-схему алгоритма нахождения факториала:



Вот код, реализующий этот алгоритм:

```
let value = prompt('Введите число');
let n = 2;
let result = 1;

while (n <= value) {
  result = result * n;
  n = n + 1;
}

console.log('Результат: ' + result);
```

Вне зависимости от того, какое число ввёл пользователь, переменная *n*, отвечающая за количество итераций в цикле **while** равна 2. Это можно объяснить тем, что умножать *result* на 1 не имело бы смысла, это пустая операция, поэтому мы начинаем сразу с двойки. Переменная *result* изначально равна 1, это минимальный результат который мы можем вернуть (вспоминаем условие, что факториал 0 = 1).

Ответим на вопрос сколько раз выполнится цикл while, если пользователь ввёл число 3:

1. В первую итерацию условие в while выполняется: 2 <= 3, соответственно, *result* умножается на 2 и становится равным 2, а переменная *n* увеличивается на 1 и становится равной 3;
2. Во вторую итерацию условие цикла while тоже выполняется: 3 <= 3, соответственно *result* (который уже равен 2) умножается на 3 и становится равным 6, а переменная *n* увеличивается на 1 и становится равной 4;

3. Последующей итерации не происходит, цикл заканчивается, т.к. не соблюдается условие: 4 <= 3 возвращает false.

Цикл завершился и в переменной *result*, как мы вычислили, хранится значение 6, его мы и выводим пользователю и это является правильным ответом.

Если пользователь ввёл 1 или 0, тело цикла не выполнится ни разу, потому что условие выполнения цикла не будет соблюдено: 2 <= 1 и 2 <= 0 возвращают false. Тем не менее, пользователю всё-равно будет возвращён правильный ответ: 1, т.к. с это значение находится в *result* изначально.

Применение цикла for

В то время как оператор **while** является универсальным и позволяет решить любую задачу, где требуется цикличность, существуют и другие операторы цикла. **do while** мы уже рассмотрели, теперь мы рассмотрим оператор цикла **for**. Решим задачу нахождения факториала с помощью него:

```
let value = prompt('Введите число');
let result = 1;

for (let n = 2; n <= value; n = n + 1) {
  result = result * n;
}

console.log('Результат: ' + result);
```

Интуитивно можно догадаться как работает оператор **for**, но давайте разберёмся. У оператора **for** есть три параметра, каждый из которых отделяется от другого точкой с запятой: for (1; 2; 3). Вот как эти параметры работают:

1. Инициализация. Этот код будет выполнен перед началом цикла в любом случае. Здесь можно объявить переменную и даже не одну. В нашем случае тут удобно объявить переменную, отвечающую за количество итераций;
2. Условие, которое проверяется перед началом каждой итерации. Если это условие не выполняется, выполнение цикла заканчивается. Этот параметр — полный аналог параметра оператора **while**;
3. Операция, которая выполняется **после** каждой итерации.

Любой из параметров может быть пустым, в самом крайнем случае корректной будет запись for (;;), это аналог while (true), то есть, объявление бесконечного цикла.

Шахматная доска

Применим полученные знания для рисования узора «шахматная доска». В нашей доске, в отличие от настоящей, будет всего 9 полей, ширина 3 поля и высота 3 поля.

Создадим цикл, который будет отвечать за рисование «строк» шахматной доски, оставим его пока пустым:

```
for (let y = 0; y < 3; y = y + 1) {
  console.log(y);
}
```

Разберём как будет выполняться этот цикл. Всё начнётся со значения y = 0, условие y < 3 будет соблюдено, в консоль будет выведено число 0. Далее выполнится y = y + 1, после чего *y* станет равным 1. В следующую итерацию условие y < 3 будет соблюдено, в консоль будет выведено число 1. Далее *y* снова увеличится на единицу и станет равным 2. В следующую итерацию условие y < 3 тоже будет соблюдено, в консоль будет выведено число 2, а *y* снова увеличится на единицу и станет равным 3. После этого условие y < 3 уже не будет выполняться и выполнение цикла завершится.

В консоль будут выведены числа 0, 1, 2, то есть, итерация выполнится 3 раза, как нам и нужно. Пусть вас не смущает нумерация с 0, а не с 1, в программировании так принято и это само по себе является поводом для шуток.

Продолжим написание кода, вложим в цикл, отвечающий за отрисовку «строки» доски, цикл, который будет отвечать за отрисовку отдельного квадрата:

```
for (let y = 0; y < 3; y = y + 1) {
  console.log('Начало выполнения вложенного цикла');
  for (let x = 0; x < 3; x = x + 1) {
    console.log('y = ' + y + ', x = ' + x);
  }
}
```

Как видно, циклы, так же как и условия, можно вкладывать друг в друга, это обычная практика. Логика выполнения верхнего цикла такая же, как мы описали её выше, только вместо вывода в консоль тут выполняется другой цикл и тоже три раза. В этом легко убедиться, если посмотреть на вывод программы:

```
> Начало выполнения вложенного цикла
> y = 0, x = 0
> y = 0, x = 1
> y = 0, x = 2
> Начало выполнения вложенного цикла
> y = 1, x = 0
```

```
> y = 1, x = 1
> y = 1, x = 2
> Начало выполнения вложенного цикла
> y = 2, x = 0
> y = 2, x = 1
> y = 2, x = 2
```

Нужно определить цвет для каждого квадрата, сделаем это таким образом:

```
let isBlack = true;
for (let y = 0; y < 3; y = y + 1) {
  console.log('Начало выполнения вложенного цикла');
  for (let x = 0; x < 3; x = x + 1) {
    let color = 'white';
    if (isBlack) {
      color = 'black';
      isBlack = false;
    } else {
      isBlack = true;
    }
    console.log('y = ' + y + ', x = ' + x + ', color = ' + color);
  }
}
```

Мы поступили самым простым способом: ввели переменную isBlack, на основе которой с помощью условий определяем значение цвета. По-умолчанию переменная color = ‘white’, но если булево значение isBlack равно true, мы переписываем значение color = ‘black’. Кроме того, мы меняем значение isBlack на противоположное каждую итерацию, чтобы обеспечить переключение цвета. Получаем нужный нам результат:

```
> Начало выполнения вложенного цикла
> y = 0, x = 0, color = black
> y = 0, x = 1, color = white
> y = 0, x = 2, color = black
> Начало выполнения вложенного цикла
> y = 1, x = 0, color = white
> y = 1, x = 1, color = black
> y = 1, x = 2, color = white
> Начало выполнения вложенного цикла
> y = 2, x = 0, color = black
> y = 2, x = 1, color = white
> y = 2, x = 2, color = black
```

Когда заготовка в виде нужных циклов, которые позволяют генерировать нужные значения для рисования шахматного узора, готова, остаётся добавить уже знакомый нам drawRect для получения нужного результата:

```
let isBlack = true;
for (let y = 0; y < 3; y = y + 1) {
  for (let x = 0; x < 3; x = x + 1) {
    let color = 'white';
    if (isBlack) {
      color = 'black';
      isBlack = false;
    } else {
      isBlack = true;
    }
    drawRect(x * 50, y * 50, 50, 50, color);
  }
}
```

Мы взяли ширину квадрата 50, а координаты просто умножаем на это значение, потому что позиция рисования каждого следующего квадрата должна отличаться от позиции рисования предыдущего на 50, а не на 1.

Не забывайте, что функция drawRect не является стандартной для JavaScript, мы добавили её в тренажёр специально для иллюстрации того, как программирование позволяет работать с графикой.

Домашнее задание

Необходимо написать программу, которая вычисляет сложные проценты по банковскому вкладу и выводит пользователю результат, детализированный по месяцам. Пользователь вводит следующие данные: размер вклада, годовой процент (без знака %), кол-во месяцев. Пример расчёта для вклада в 1000 рублей под 6% годовых на три месяца:

1. Первый месяц: 1000 + 1000 * (6% / 100 / 12) = 1005
2. Второй месяц: 1005 + 1005 * (6% / 100 / 12) = 1010.025
3. Третий месяц: 1010.025 + 1010.025 * (6% / 100 / 12) = 1015,07

Итого доход: 15.07

Дополнительно

Написать программу, которая рисует доску для игры «Крестики-нолики».

Глоссарий

- Цикл — разновидность управляющей конструкции, предназначенная для организации многократного исполнения набора инструкций;
- Итерация — повторение какого-либо действия;
- Бесконечный цикл — цикл, реализованный таким образом, что условие выхода из него никогда не выполняется;
- Сложные проценты (капитализация процентов) — причисление процентов к сумме вклада, позволяет в дальнейшем осуществлять начисление процентов на проценты путем выполнения двойной операции — выплата процентов и пополнение;
- Целое число — расширение множества натуральных чисел, получаемое добавлением к нему нуля и отрицательных чисел;
- Дробное (рациональное) число — число, которое можно представить обыкновенной дробью m/n;
- Округление — замена числа на его приближённое значение (с определённой точностью), записанное с меньшим количеством значащих цифр.

Дополнительные материалы

- Цикл for на MDN — https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Statements/for;

Используемые источники

- https://wikipedia.org