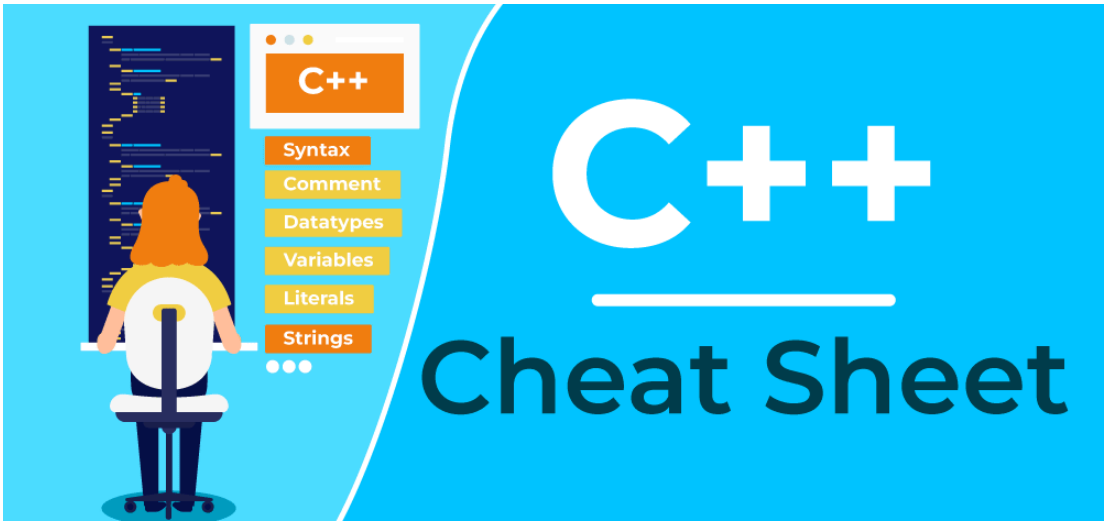
Поиск...

Шпаргалка по C++

Последнее обновление: 01 мая 2025 г.

Это шпаргалка по программированию на C++. Она полезна для новичков и продвинутых пользователей, желающих изучить или повторить концепции программирования на C++. При изучении нового языка раздражает необходимость переключаться между страницами и искать разные веб-сайты для разных понятных концепций. С помощью этой шпаргалки вы можете очень легко изучить концепции C++ .



C++ — язык программирования высокого уровня. Разработан в 1983 году Бьярном Страуструпом в Bell Labs. Используется для разработки различных приложений.

Базовая структура программ на языке C++

```cpp
// Header files
#include<bits/stdc++.h>

// std namespace contains
// various standard library components
using namespace std;

// main function is the starting
//point of program execution
int main() {

    // This is the section where
    // we write code statements
    return 0;
}
```

Комментарии

Комментарии в C++ используются для пояснения кода, что облегчает другим понимание функциональности кода. Они не выполняются компилятором. Комментарии также могут использоваться для временного отключения определенных операторов кода без их удаления.

Существует два типа комментариев:

1. Однострочный

Мы используем две косые черты // для обозначения однострочного комментария. Например,

```
// This is a comment
```

2. Многострочный

Мы используем /* для начала многострочного комментария и */ для его завершения. Например,

```
/* This is multi-lined comment.
The below statement will print GeeksforGeeks.*/
```

Переменные

В C++ **переменная** — это контейнер, используемый для хранения значений данных:

- Перед использованием переменные должны быть объявлены.
- Несколько переменных также могут быть объявлены одновременно.
- Имя переменной может содержать буквы, цифры и символ подчеркивания, но имя переменной должно начинаться с буквы или символа подчеркивания.

***Идентификаторы** : Всем переменным в программе должны быть присвоены уникальные имена, известные как идентификаторы, чтобы каждую переменную можно было однозначно идентифицировать.*

***Константы** : Константы — это фиксированные значения, которые остаются неизменными во время выполнения программы.*

Синтаксис

```
// Declaring a single variable
data_type var_name;

// Declaring multiple variables
data_type var1_name, var2_name, var3_name;
```

Типы данных

Типы данных — это тип данных, которые переменная может хранить в программе.

1. Целое число

- Используется для хранения целых чисел.
- Целые числа занимают 4 байта памяти.

Пример

```
int var = 123;
```

2. Характер

- Используется для хранения символов.
- Занимает 1 байт памяти.

Пример

```
char var = 'a';
```

3. Плавающая точка

- Используется для хранения чисел с плавающей запятой одинарной точности.
- Занимает 4 байта памяти.

Пример

```
float num = 1.23;
```

4. Двойной

- Используется для хранения чисел с плавающей запятой двойной точности.
- Занимает 8 байт памяти.

Пример

```
double num = 1.2345;
```

5. Булевский

- Он используется для хранения логических значений, которые могут быть либо истинными, либо ложными.

Пример

```
bool b = false;
```

6. Струна

- Строка — это набор символов , заключенных в двойные кавычки. Тип данных string используется для хранения слов или предложений.
- Строковый тип данных является частью стандартной библиотеки и определен в заголовочном файле **<string>** .
- Для использования класса string нам необходимо включить заголовочный файл **<string> .**

Пример

```
string str = "GeeksforGeeks";
```

Ввод и вывод

1. Ввод от пользователя: Мы можем получить ввод от пользователя, используя **cin** из библиотеки **iostream** . Например,

```
int var;
cin >> var;
```

2. Вывод на консоль: Мы можем вывести вывод на консоль, используя **cout** из библиотеки **iostream** . Например,

```
cout << “Hello World”;
```

Новые Линии

Мы можем использовать символ **\n или endl** для вставки новой строки. Например,

```
cout << "Hello World! \n";
cout << "Hello World!" << endl;
```

Условные утверждения

Условные операторы позволяют нам контролировать ход программы на основе определенных условий. Они помогают нам запускать определенный раздел кода на основе условия.

1. Оператор if

Оператор If выполняет блок кода тогда и только тогда, когда заданное условие истинно.

Синтаксис

```
if (condition) {
    // Code to be executed if the condition is true
}
```

2. Вложенный оператор if

Синтаксис

```
if (condition1) {
    // Code to be executed
    if (condition2) {
        // Code to be executed
    }
}
```

3. Оператор if-else

В **операторе if-else** условие внутри оператора **if** истинно, тогда код внутри блока if будет выполнен, в противном случае будет выполнен код внутри блока else.

Синтаксис

```
if (condition) {
    // Code to be executed
    // if the condition is true
} else {
    // Code to be executed
    // if the condition is false
}
```

4. Оператор else-if

Оператор else if позволяет последовательно проверять несколько условий.

Синтаксис

```
if (condition1) {
    // Code to be executed
    // if condition1 is true
} else if (condition2) {
    // Code to be executed if
    // condition1 is false and
    // condition2 is true
} else {
    // Code to be executed if
    // all conditions are false
}
```

5. Сокращенная запись if else (тернарный оператор)

Сокращенная запись if else, также известная как **тернарный оператор (?:)**, работает так же, как операторы if-else, которые можно использовать для сокращения количества строк кода.

Синтаксис

```
(condition) ? expression1 : expression2;
```

Условие **внутри круглых скобок ()** истинно, выражение1 будет вычислено и станет результатом выражения. В противном случае, если условие ложно, выражение2 будет вычислено и станет результатом.

6. Оператор переключения

Оператор **switch** оценивает выражение и сравнивает значение выражения с case. Если выражение соответствует значению любого из case, будет выполнен код, связанный с этим case.

7. Разрыв и дефолт

Ключевое слово **break** используется для выхода из оператора switch, когда один из вариантов совпадает, в то время как ключевое слово **default** , которое является необязательным, выполняется, когда ни один из вариантов не соответствует значению выражения.

Синтаксис

```
switch (expression) {
    case value1:
        // Code to be executed if
        // expression matches value1
        break;
    case value2:
        // Code to be executed if
        // expression matches value2
        break;
    // ...
    default:
        // Code to be executed if
        // expression does not match any case
        break;
}
```

> ***Примечание:*** *ключевые слова по умолчанию используются с операторами switch.*

Петли

Циклы используются для многократного выполнения блока кода.

Типы петель

1. Цикл For

Цикл For помогает нам выполнить блок кода фиксированное количество раз.

Синтаксис:

```
for (initialization expr; test expr; update expr) {
    // body of the for loop
}
```

2. Цикл While

Цикл While многократно выполняет блок кода до тех пор, пока заданное условие не станет истинным.

Синтаксис

```
while (condition) {
   // statements
   update_condition;
}
```

3. Цикл Do-While

Цикл do-while также выполняет блок кода до тех пор, пока условие не станет истинным, но разница между циклом while и циклом do-while заключается в том, что цикл do-while выполняет код один раз, не проверяя условие, а тестовое условие проверяется в конце тела цикла.

Синтаксис

```
do {
    // Body of do-while loop
} while (condition);
```

Массивы

Массив — это структура данных , которая позволяет хранить фиксированное количество элементов одного типа данных в смежных ячейках памяти.

Синтаксис:

```
dataType array_name[size];
```

где,

- **data_type** : Тип данных, которые будут храниться в массиве.
- **array_name** : Имя массива.
- **размер** : Размер массива.

Пример

```cpp
#include <iostream>
#include <string>
using namespace std;
int main() {

    // Declare and initialize an array of strings
    string fruits[] = {"Apple", "Banana", "Orange", "Grapes"};

    // Access and print each element of the array
    for (int i = 0; i < 4; i++) {
        cout << "Fruit at index " << i << ": " << fruits[i] << endl;
    }

    return 0;
}
```

Выход

```
Фрукт с индексом 0: Яблоко
Фрукт с индексом 1: Банан
Фрукт с индексом 2: Апельсин
Фрукт с индексом 3: Виноград
```

Многомерные массивы

Многомерные массивы известны как массивы массивов, которые хранят схожие типы данных в табличной форме.

Синтаксис:

```
data_type array_name[size1][size2]....[sizeN];
```

где **size1, size2,..., sizeN** — размеры каждого измерения.

Двумерные массивы являются наиболее часто используемыми многомерными массивами в C++.

Пример

```cpp
#include <iostream>
using namespace std;
int main()
{
    // Declaration and initialization of a 2D array
    int arr[3][4] = { { 1, 2, 3, 4 },
                      { 5, 6, 7, 8 },
                      { 9, 10, 11, 12 } };

    // Accessing elements in the 2D array
    // Output: 1
    cout << "Element at arr[0][0]: " << arr[0][0] << endl;
```

```
    // Output: 7
    cout << "Element at arr[1][2]: " << arr[1][2] << endl;

    // Changing the value of an element
    // Output: 20
    arr[2][3] = 20;
    cout << "Modified element at arr[2][3]: " << arr[2][3]
         << endl;

    // Nested loops for iterating through the 2D array
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 4; j++) {
            cout << arr[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```

Выход

```
Элемент в arr[0][0]: 1
Элемент в arr[1][2]: 7
Измененный элемент в arr[2][3]: 20
1 2 3 4
5 6 7 8
9 10 11 20
```

Векторы

Векторы — это динамическая структура данных, подобная массиву, в которой элементы одного типа данных хранятся в непрерывном порядке и могут автоматически изменять свой размер, в отличие от массивов, которые означают, что векторы могут увеличиваться при вставке элемента или уменьшаться при его удалении.

- Векторы присутствуют в стандартной библиотеке шаблонов C++ (STL).
- Для использования векторов нам необходимо включить заголовочный файл <vector> в нашу программу на C++.

Синтаксис:

```
vector<data_type> vector_name;
```

Часто используемые векторные функции

- *push_back() – используется для вставки элементов в конец вектора.*
- ***pop_back()** – используется для извлечения или удаления элементов из конца вектора.*
- ***clear()** – используется для удаления всех элементов вектора.*
- ***empty()** – используется для проверки того, является ли вектор пустым.*

- ***at(i)*** *– используется для доступа к элементу по указанному индексу «i».*
- ***front()*** *– используется для доступа к первому элементу вектора.*
- ***back()*** *– используется для доступа к последнему элементу вектора.*
- ***erasure()*** *– используется для удаления элемента в указанной позиции.*

Пример:

```cpp
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    // Create an empty vector
    vector<int> numbers;

    // push_back()
    numbers.push_back(10);
    numbers.push_back(20);
    numbers.push_back(30);

    // Accessing elements using at()
    // Output: 10
    cout << "Element at index 0: " << numbers.at(0) << endl;
    // Output: 20
    cout << "Element at index 1: " << numbers.at(1) << endl;

    // front() and back()
    // Output: 10
    cout << "First element: " << numbers.front() << endl;
    // Output: 30
    cout << "Last element: " << numbers.back() << endl;

    // pop_back()
    // Remove the last element
    numbers.pop_back();

    // erase()
    // Remove the element at index 1
    numbers.erase(numbers.begin() + 1);

    // empty()
    if (numbers.empty()) {
        cout << "Vector is empty" << endl;
    }
    else {
        cout << "Vector is not empty" << endl;
    }

    // clear()
    // Remove all elements
    numbers.clear();

    if (numbers.empty()) {
        cout << "Vector is empty" << endl;
    }
    else {
        cout << "Vector is not empty" << endl;
    }

    return 0;
}
```

Выход

```
Элемент с индексом 0: 10
Элемент с индексом 1: 20
Первый элемент: 10
Последний элемент: 30
Вектор не пустой
Вектор пуст
```

Ссылки и указатели

Ссылки

Ссылки предоставляют псевдоним для существующей переменной. Мы можем манипулировать исходным значением, используя ссылочную переменную. Ссылочная переменная объявляется с помощьюоператора **&** .

Пример

```
int var= 12;
// A reference variable to var
int& ref= var;
```

где **ref** — ссылка на переменную **var** .

Указатели

Указатель — это переменная, которая хранит адрес памяти другой переменной. Его можно создать с помощью оператора * , а адрес другой переменной можно присвоить с помощью оператора address-of & .

Пример

```
int i = 3;
// A pointer to variable i or "stores the address of i"
int *ptr = &i;
```

Функции

Функции — это повторно используемый блок набора операторов, который выполняет определенную задачу. Функции могут использоваться для организации логики программы.

Синтаксис объявления функции

```
return_type function_name(parameters);
```

Синтаксис определения функции

```
return_type function_name(parameters) {
    // function body
```

```
    // code to be executed
    // return statement (if applicable)
}
```

- **return_type** : это тип данных значения, возвращаемого функцией.
- **function_name** : Это имя функции.
- **параметры** : параметры — это входные значения, предоставляемые при вызове функции. параметры необязательны.

Пример

Программа для сложения двух чисел.

```
#include<bits/stdc++.h>
using namespace std;

// Function declaration
int sum(int a, int b);

// Function definition
int sum(int a, int b) {
    return a + b;
}

int main()
{
    // Function call
    int result = sum(3, 4);
      cout << result;
}
```

Выход

```
7
```

Объяснение : Функция **sum** принимает два целых числа в качестве параметров и возвращает сумму двух чисел. Тип возвращаемого значения функции — **int** . Параметрами функции являются два целых числа: 3 и 4. Возвращаемое значение 7 сохраняется в переменной result.

Строковые функции

В стандартной библиотеке шаблонов в C++ есть несколько строковых функций, которые используются для выполнения операций со строками. Вот некоторые из наиболее часто используемых строковых функций:

1. Функция length()

Возвращает длину строки.

Пример

```
string str = "GeeksforGeeks";
cout << "String length: " << str.length();
```

2. Функция substr()

Используется для извлечения **подстроки** из заданной строки.

Синтаксис

```
string substr (size_t pos, size_t len) const;
```

- **pos** : Позиция первого символа для копирования.
- **len** : Длина подстроки.
- **size_t** : Это беззнаковый целочисленный тип.

Пример

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
    string str = "GeeksforGeeks";

    // Extracts a substring starting from
    // index 1 with a length of 5
    string sub = str.substr(1, 5);

    cout << "Substring: " << sub << endl;

    return 0;
}
```

Выход

```
Подстрока: eeksf
```

3. Функция append()

Добавляет строку в конец заданной строки.

Пример

```
#include <iostream>
#include <string>
using namespace std;
int main() {
    string str = "Geeksfor";

    str.append("Geeks");

    cout << "Appended string: " << str << endl;

    return 0;
}
```

Выход

```
Добавленная строка: GeeksforGeeks
```

4. Функция сравнения()

Он используется для лексикографического **сравнения** двух строк.

Пример

```cpp
#include <iostream>
#include <string>
using namespace std;
int main() {
    string str1 = "Geeks";
    string str2 = "for";
    string str3 = "Geeks";

    int result1 = str1.compare(str2);
    cout << "Comparison result: " << result1 << endl;

    int result2 = str1.compare(str3);
    cout << "Comparison result: " << result2 << endl;

    return 0;
}
```

Выход

```
Результат сравнения: -31
Результат сравнения: 0
```

5. Функция empty()

Используется для проверки того, является ли строка **пустой** .

Пример

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    string str1 = "GeeksforGeeks";
    string str2 = "";

    if (str1.empty())
        cout << "str1 is empty" << endl;
    else
        cout << "str1 is not empty" << endl;

    if (str2.empty())
        cout << "str2 is empty" << endl;
    else
        cout << "str2 is not empty" << endl;
    return 0;
}
```

Выход

```
str1 не пуста
str2 пуст
```

Математические функции

Функция	Описание	Пример
мин(x, y)	Возвращает минимальное значение x и y.	cout << min(10, 20);
макс(x, y)	Возвращает максимальное значение x и y.	cout << макс(10, 20);
квадратный корень(x)	Возвращает квадратный корень x.	cout << sqrt(25);
ceil(x)	Он округляет значение x до ближайшего целого числа.	double ceilX = ceil(3.14159);
пол(x)	Он округляет значение x в меньшую сторону до ближайшего целого числа.	двойной полX = пол(3.14159);
pow(x,n)	Возвращает значение x, возведенное в степень y.	двойной результат = pow(3.0, 2.0);

Объектно-ориентированное программирование

Объектно-ориентированное программирование обычно подразумевает хранение данных в форме классов и объектов.

Класс и объекты

- Класс: Класс — это определяемый пользователем тип данных, содержащий его элементы данных и функции-члены. Класс — это план для объектов, имеющих схожие атрибуты и поведение.
- **Объекты:** Объект — это экземпляр или переменная класса.

Столпы ООП

1. Инкапсуляция

Инкапсуляция — это упаковка данных и методов в единое целое. В C++ для инкапсуляции используются классы.

2. Абстракция

Показ только необходимых деталей и сокрытие внутренних деталей называется **абстракцией** .

4. Наследование

Выведение свойств класса (родительского класса) к другому классу (дочернему классу) называется наследованием. Оно используется для повторного использования кода.

Типы наследования:

- **Одиночное наследование** : когда производный класс наследует свойства одного базового класса, это называется одиночным наследованием.
- **Множественное наследование** : когда производный класс наследует свойства нескольких базовых классов, это называется множественным наследованием.
- **Многоуровневое наследование** : когда производный класс наследует свойства другого производного класса, это называется многоуровневым наследованием.
- **Иерархическое наследование** : когда несколько производных классов наследуют свойства одного базового класса, это называется иерархическим наследованием.
- **Гибридное (виртуальное) наследование** : Когда мы объединяем более одного типа наследования, это называется гибридным (виртуальным) наследованием. Пример: объединение многоуровневого и иерархического наследования.

3. Полиморфизм

Предоставление различных функциональных возможностей функциям или операторам с одинаковым именем известно как **полиморфизм** .

C++ предоставляет два типа полиморфизма:

- **Полиморфизм во время компиляции** может быть достигнут с помощью:
    - Перегрузка оператора
    - Перегрузка функций

- **Полиморфизм во время выполнения** может быть достигнут с помощью:
    - Переопределение функции
    - Виртуальные функции

4. Наследование

Выведение свойств класса (родительского класса) к другому классу (дочернему классу) называется **наследованием** . Оно используется для повторного использования кода.

Типы наследования:

- **Одиночное наследование** : когда производный класс наследует свойства одного базового класса, это называется одиночным наследованием.
- **Множественное наследование** : когда производный класс наследует свойства нескольких базовых классов, это называется множественным наследованием.
- **Многоуровневое наследование** : когда производный класс наследует свойства другого производного класса, это называется многоуровневым наследованием.
- **Иерархическое наследование** : когда несколько производных классов наследуют свойства одного базового класса, это называется иерархическим наследованием.
- **Гибридное (виртуальное) наследование** : Когда мы объединяем более одного типа наследования, это называется гибридным (виртуальным) наследованием. Пример: объединение многоуровневого и иерархического наследования.

Обработка файлов

Обработка файлов означает чтение данных из файла и манипулирование данными файла.

Операции по обработке файлов

*1. **Открытие файла** : мы можем использовать функцию-член **open()** **класса ofstream** , чтобы открыть файл.*

*2. **Чтение файла** : мы можем использовать функцию-член **getline()** **класса ifstream** для чтения файла.*

*3. **Запись в файл** : мы можем использовать оператор << для записи в файл после открытия файла с помощью объекта класса **ofstream** .*

Пример:

```cpp
#include <fstream>
#include <iostream>
#include <string>

using namespace std;

int main()
{
    ofstream outputFile("example.txt");

    // Open the file for writing
    outputFile.open("example.txt");
    if (outputFile.is_open()) {

        // Write data to the file
        outputFile << "Hello, World!" << endl;
        outputFile << 42 << endl;
        outputFile.close(); // Close the file
    }
    else {

        // Failed to open the file
        cout << "Error opening the file for writing."
             << endl;
        return 1;
    }

    // Reading from a file
    ifstream inputFile("example.txt");
    if (inputFile.is_open()) {
        string line;
        while (getline(inputFile, line)) {
            // Print each line
            cout << line << endl;
        }
        // Close the file
        inputFile.close();
    }
    else {

        // Failed to open the file
        cout << "Error opening the file for reading."
             << endl;
        return 1;
    }

    return 0;
}
```

Эта шпаргалка по C++ может служить справочным руководством для программистов, обеспечивающим быстрый доступ к концепциям C++.

Следующая статья

Шпаргалка по языку C

Похожие чтения

Geeksforgeeks Cheatsheets - Все коллекции шпаргалок по кодированию

Шпаргалки — это короткие документы, которые содержат всю самую важную информацию о конкретной технологии вкратце, например, ее синтаксис, команды, функции или ее особенности. Шпаргалки...

4 мин чтения

Памятка по маске подсети

Маска подсети — это числовое значение, которое описывает, как компьютер или устройство разделяет IP-адрес на две части: сетевую часть и хостовую часть. Сетевой элемент определяет сеть, к которой...

9 мин чтения

Шпаргалка по Git

Git Cheat Sheet — это всеобъемлющее краткое руководство по изучению концепций Git, от самых базовых до продвинутых уровней. С помощью этой Git Cheat Sheet мы стремимся предоставить удобный справочный...

10 мин чтения

Шпаргалка по NumPy: от новичка до продвинутого пользователя (PDF)

NumPy означает Numerical Python. Это один из важнейших основополагающих пакетов для численных вычислений и анализа данных в Python. Большинство вычислительных пакетов, предоставляющих научные...

15+ мин чтения

Шпаргалка по командам Linux

Linux, часто ассоциируемый с тем, что он является сложной операционной системой, используемой в основном разработчиками, не обязательно полностью соответствует этому описанию. Хотя поначалу он...

13 мин чтения

Памятка Pandas по науке о данных на Python

Pandas — это мощная и универсальная библиотека, которая позволяет работать с данными в Python. Она предлагает ряд функций и возможностей, которые делают анализ данных быстрым, простым и эффективны...

15+ мин чтения

Шпаргалка по Java

Java — это язык программирования и платформа, которые широко используются с момента их разработки Джеймсом Гослингом в 1991 году. Он следует концепции объектно-ориентированного программирования и...

15+ мин чтения

Шпаргалка по C++ STL

Шпаргалка по C++ STL содержит краткие и лаконичные заметки по стандартной библиотеке шаблонов (STL) в C++. Шпаргалка по STL, разработанная для программистов, которые хотят быстро ознакомиться с ключевым...

15+ мин чтения

Шпаргалка по Docker: полное руководство (2024)

Docker — очень популярный инструмент, представленный для того, чтобы упростить разработчикам создание, развертывание и запуск приложений с использованием контейнеров. Контейнер — это утилита,...

11 мин чтения

Шпаргалка по C++

Это шпаргалка по программированию на C++. Она полезна для новичков и продолжающих, желающих изучить или повторить концепции программирования на C++. При изучении нового языка раздражает...

15+ мин чтения

CSS
JavaScript
Машинопись
ReactJS
СледующийJS
Бутстрап
Веб-дизайн

Проекты Python
Питон Tkinter
Веб-скрапинг с помощью Python
Учебник OpenCV
Вопрос на собеседовании по Python
Джанго

Информатика

Операционные системы
Компьютерная сеть
Система управления базами данных
Программная инженерия
Проектирование цифровой логики
Инженерная математика
Разработка программного обеспечения
Тестирование программного обеспечения

DevOps

Гит
Линукс
АВС
Докер
Кубернетес
Лазурный
GCP
Дорожная карта DevOps

Проектирование системы

Проектирование высокого уровня
Низкоуровневое проектирование
Диаграммы UML
Руководство по собеседованию
Шаблоны проектирования
ООАД
Учебный лагерь по проектированию систем
Вопросы для интервью

Подготовка к собеседованию

Конкурсное программирование
Лучший DS или Алгоритм для CP
Процесс подбора персонала в компании
Подготовка на уровне компании
Подготовка к способностям
Пазлы

Школьные предметы

Математика
Физика
Химия
Биология
Социальные науки
Грамматика английского языка
Коммерция
Мировой ГК

Видео GeeksforGeeks

ДСА
Питон
Ява
C++
Веб-разработка
Наука о данных
Предметы CS