**🧠 Памятка: Условные операторы в C#**

**1. if, else if, else — базовая проверка условий**

✅ Используй, когда:

* Нужно проверить **несколько независимых условий**
* Простой линейный выбор

📌 Пример:

int age = 20;

if (age < 18) {

Console.WriteLine("Too young");

} else if (age == 18) {

Console.WriteLine("Just turned adult");

} else {

Console.WriteLine("Adult");

}

**2. Вложенные if — когда нужно проверить дополнительные условия внутри условий**

✅ Используй, когда:

* Второе условие **зависит** от первого

📌 Пример:

if (age >= 18) {

if (hasLicense) {

Console.WriteLine("Can drive");

} else {

Console.WriteLine("Get a license");

}

} else {

Console.WriteLine("Too young");

}

**3. switch — если у тебя есть одно значение, и ты сравниваешь его с множеством фиксированных вариантов**

📌 Пример:

csharp

КопироватьРедактировать

int month = 4;

string season = "";

switch (month) {

case 12:

case 1:

case 2:

season = "Winter";

break;

case 3:

case 4:

case 5:

season = "Spring";

break;

default:

season = "Unknown";

break;

}

**4. Тернарный оператор ?: — короткая форма if-else**

✅ Используй, когда:

* Нужно сделать **короткое присваивание** на основе условия

📌 Пример:

csharp

КопироватьРедактировать

int age = 16;

string status = (age >= 18) ? "Adult" : "Minor";

Можно вкладывать тернарные, если 3 варианта:

csharp

КопироватьРедактировать

string result = (num > 0) ? "positive" : (num < 0) ? "negative" : "zero";

Конечно! Вот краткий и понятный **конспект-памятку** по темам:

**🖨️ Вывод данных (Console.WriteLine и Console.Write)**

| **Метод** | **Описание** |
| --- | --- |
| Console.WriteLine | Выводит текст и переносит курсор на новую строку |
| Console.Write | Выводит текст без переноса строки |

**Пример:**

Console.WriteLine("Hello"); // Выведет: Hello

Console.Write("World"); // Выведет: World (на той же строке)

**🧵 Интерполяция строк (вставка значений внутрь строки)**

Интерполяция — это способ вставить переменные прямо внутрь строки с помощью знака $.

**Пример:**

string name = "Alice";

int age = 25;

Console.WriteLine($"Name: {name}, Age: {age}");

// Выведет: Name: Alice, Age: 25

Также можно использовать сложение строк:

Console.WriteLine("Name: " + name + ", Age: " + age);

Но **интерполяция читается проще и короче**.

в C# ты можешь использовать **многострочную интерполяцию строк** (так называемый *verbatim string*) с помощью @ и $ — чтобы вывести сразу несколько строк с подставленными значениями, **в одном Console.WriteLine**.

Вот как можно переписать твой вывод:

КопироватьРедактировать

Console.WriteLine($@"

Sum: {sum}

Difference: {difference}

Product: {product}

Quotient: {quotient}");

🔹 Обрати внимание на $@ — это сочетание интерполяции ($) и многострочного литерала (@).

**🧾 Памятка: Форматированный вывод в C#**

**📌 Что это?**

Форматированный вывод позволяет управлять тем, **как именно** числа, даты и другие значения отображаются на экране.

**🔹 Форматирование через интерполяцию строк**

double num = 123.456;

Console.WriteLine($"Value: {num:F2}");

✅ Результат:

Value: 123.46

:F2 — означает формат *Fixed-point* (с фиксированным количеством знаков после запятой), в данном случае **2 знака**.

**🔹 Другие форматы:**

|  |  |  |
| --- | --- | --- |
| Формат | Пример | Что делает |
| F2 | 123.456 → 123.46 | Фиксированное количество знаков после запятой |
| N0 | 1234567.89 → 1,234,568 | Число с разделением тысяч, без дробной части |
| P | 0.75 → 75.00 % | Формат процента |
| C | 123.45 → $123.45 | Денежный формат (учитывает локаль) |
| E | 1234 → 1.234000E+003 | Экспоненциальная форма |

**🔹 Альтернатива: string.Format**

double num = 123.456;

Console.WriteLine(string.Format("Value: {0:F2}", num));

То же самое, но полезно, когда нужно составить строку **без интерполяции**.

**🔹 Пример форматирования нескольких значений**

double a = 10.1234;

double b = 2.5;

Console.WriteLine($"Sum: {a + b:F2}");

Console.WriteLine($"Difference: {a - b:F2}");

Console.WriteLine($"Product: {a \* b:F2}");

Console.WriteLine($"Quotient: {a / b:F2}");

📤 Вывод:

Sum: 12.62

Difference: 7.62

Product: 25.31

Quotient: 4.05

**📥 Ввод данных (Console.ReadLine)**

Console.ReadLine() считывает строку, введённую пользователем в консоли.

**Пример:**

string input = Console.ReadLine();

**🔄 Преобразование строки в число (Parse)**

Методы int.Parse, double.Parse, bool.Parse преобразуют введённую строку в нужный тип:

| **Метод** | **Назначение** |
| --- | --- |
| int.Parse() | Строка → целое число |
| double.Parse() | Строка → число с точкой |
| bool.Parse() | Строка → булевое значение |

**Примеры:**

int number = int.Parse(Console.ReadLine()); // Ввод: 42

double price = double.Parse(Console.ReadLine()); // Ввод: 3.14

bool hasAccess = bool.Parse(Console.ReadLine()); // Ввод: true или false

⚠️ Если пользователь введёт неверное значение (например, "abc" вместо числа), программа **упадёт с ошибкой**. Чтобы избежать этого, позже изучают TryParse.

**🔁 Циклы в C#**

Циклы позволяют выполнять блок кода многократно, пока выполняется условие.

**1. for**

**Когда использовать:**  
Когда известно количество итераций.

**Синтаксис:**

for (int i = 0; i < 5; i++) {

Console.WriteLine(i);

}

**Объяснение:**

* i = 0 — начальное значение
* i < 5 — условие цикла
* i++ — шаг итерации

**2. while**

**Когда использовать:**  
Когда количество итераций заранее неизвестно, и нужно продолжать, пока выполняется условие.

**Синтаксис:**

int i = 0;

while (i < 5) {

Console.WriteLine(i);

i++;

}

**3. do-while**

**Когда использовать:**  
Когда нужно гарантировать хотя бы одну итерацию, даже если условие изначально ложно.

**Синтаксис:**

int i = 0;

do {

Console.WriteLine(i);

i++;

} while (i < 5);

**4. break**

**Что делает:**  
Прерывает выполнение цикла немедленно.

**Пример:**

for (int i = 0; i < 10; i++) {

if (i == 5) {

break; // остановится на 5

}

Console.WriteLine(i);

}

**5. continue**

**Что делает:**  
Пропускает текущую итерацию и переходит к следующей.

**Пример:**

for (int i = 0; i < 5; i++) {

if (i == 2) {

continue; // 2 не будет напечатано

}

Console.WriteLine(i);

}

**6. Вложенные циклы**

**Когда использовать:**  
Когда нужно выполнить один цикл внутри другого, например, для перебора пар значений.

**Пример:**

for (int i = 1; i <= 3; i++) {

for (int j = 1; j <= 2; j++) {

Console.WriteLine($"i: {i}, j: {j}");

}

}

**7. Бесконечный цикл**

**Когда использовать:**  
Когда цикл должен выполняться вечно (например, в игровых циклах или серверах), обычно с break.

**Примеры:**

while (true) {

string input = Console.ReadLine();

if (input == "exit") break;

}

Или:

for (;;) {

// Бесконечно

}

Конечно! Вот расширенная шпаргалка по методам в C# с пояснением, **когда и зачем использовать** каждую конструкцию:

**📌 C# Methods (Функции) — Расширенная шпаргалка**

**✅ 1. Declaring Methods (Объявление методов)**

**Когда использовать:**  
Когда у тебя есть **блок кода**, который ты **хочешь переиспользовать** несколько раз или просто **логически отделить** от Main.

**Синтаксис:**

public static ReturnType MethodName([параметры]) {

// действия

return значение; // если return type не void

}

**Пример:**

public static void ShowWelcome() {

Console.WriteLine("Добро пожаловать!");

}

**🎯 2. Method Parameters (Параметры методов)**

**Когда использовать:**  
Когда метод должен работать **с входными данными**, отличающимися каждый раз.

**Пример:**

public static void GreetUser(string name) {

Console.WriteLine("Привет, " + name);

}

**Преимущества:**

* Гибкость: один метод — много вариантов использования
* Повышает читаемость и повторное использование

**🔄 3. Return Types (Возвращаемое значение)**

**Когда использовать:**  
Когда метод **должен вернуть результат** своей работы (например, сумму, максимум, проверку и т.д.).

**Пример:**

public static int Add(int a, int b) {

return a + b;

}

**Когда лучше использовать void:**  
Если результат не нужен, а метод делает, например, только вывод (Console.WriteLine) или изменение состояния.

**🧩 4. Optional Parameters (Необязательные параметры)**

**Когда использовать:**  
Когда ты хочешь сделать **часть параметров необязательной**, особенно если есть часто повторяющееся значение по умолчанию.

**Пример:**

public static void Welcome(string name = "Гость") {

Console.WriteLine("Привет, " + name);

}

**Преимущество:**

* Меньше перегрузок методов
* Более читаемый код

**✅ 5. Validation Function (Проверяющая функция)**

**Когда использовать:**  
Когда тебе нужно проверить данные и вернуть **true/false**. Это классический пример **булевой логики**.

**Пример:**

public static bool IsValidLogin(string username, string password) {

if (username == "admin") return true;

return username == "user" && password == "qweasd";

}

**Применение:**

* Авторизация
* Проверка условий
* Фильтрация данных

**❌ 6. Void Methods (Методы без return)**

**Когда использовать:**  
Когда метод **не должен возвращать результат**, а просто выполняет действия — например:

* Вывод на экран
* Изменение глобального состояния
* Логирование

**Пример:**

public static void PrintResult(int res) {

Console.WriteLine("Результат: " + res);

}

**🛠 Когда использовать что?**

| **Задача** | **Использовать** |
| --- | --- |
| Печать, вывод, логика без результата | void метод |
| Нужно вернуть значение | Метод с return и нужным типом |
| Нужен гибкий ввод | Параметры метода |
| Значение может быть необязательным | Необязательные параметры (=) |
| Повторяющийся код | Вынести в метод |
| Проверка (логика true/false) | Метод с bool и return |

**💡 Дополнительно:**

* **Методы делают код чище** — меньше дублирования, понятные блоки.
* **Хорошее имя метода** должно отвечать на вопрос «что делает эта функция?»
* **Пиши короткие методы.** Лучше 3 маленьких, чем 1 большой.