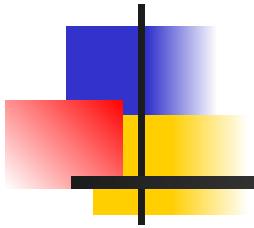
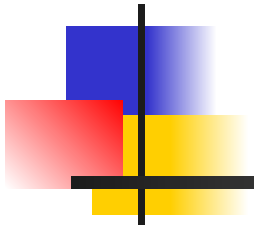


Модуль 2

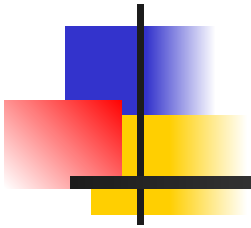
Переменные и типы данных



Обзор модуля

В этом модуле вы изучите:

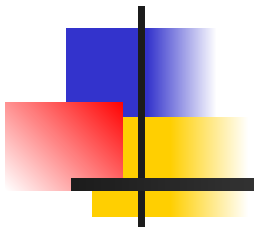
- Переменные и типы данных в C#
- Комментарии и XML-документацию
- Константы и литералы
- Ключевые слова и Escape-последовательности
- Ввод и вывод



Занятие 1 - Переменные и типы данных в C#

На первом занятии, **Переменные и типы данных в C#**, вы научитесь:

- Описывать переменные и их цели.
- Описывать типы данных и их цели.
- Определять базовые типы данных в C#.
- Описывать ссылочные типы данных.
- Объяснять правила именования переменных.
- Объявлять и использовать переменные.



Определение

- **Переменная -** это сущность, значение которой может изменяться. Например, возраст студента, зарплата сотрудника.

- В C# **переменные -** это область компьютерной памяти, которая идентифицируется уникальным именем и используется для хранения значений.

- Они базируются на типах данных, которые необходимы для хранения (переменные могут быть различных типов).

Использование переменных

- **Объявление** …енные выделяется в момент их создания.

```
<datatype> <variableName>;
string empName;
```

где,
`datatype`: допустимый в C# тип данных.
`variableName`: допустимое имя переменной.

- **Синтаксис - Инициализация** …овать переменную в момент создания, или позднее.

```
<variableName> = <value>;
```

где,
=: оператор присваивания, используемый для установки значения.
`value`: данные, хранящиеся в переменной.

- **Пример** переменной.

Типы данных

Типы значения

- Хранят актуальное значение в стеке
- Значения могут быть любого встроенного или определенного пользователем типа данных
- Значения встроенных типов данных □ int, float, double, char и bool

Ссылочные типы

- Хранят адреса размещения в памяти других переменных
- Значения могут принадлежать любому встроенному или пользовательскому типу данных
- Большинство определенных пользователем типов, например, классы, являются ссылочными типами

Компилятор должен знать, какой тип данных хранит конкретная переменная.

Это позволяет переменным хранить значения соответствующих типов данных.

В языке программирования C# типы данных делятся на две категории. Это:

- Типы-значения
- Ссылочные типы

Предопределённые типы данных

Тип данных	Размер	Диапазон
byte	Беззнаковое 8-битное целое	От 0 до 255
short	знаковое 16-битное целое	От -32,768 до 32,767
int	знаковое 32-битное целое	От -2,147,483,648 до 2,147,483,647
long	знаковое 64-битное целое	От – 9,223,372,036,854,775,808 до 9,223,372,036,854,775,807
float	32-битное вещественное число с точностью 7 знаков	От ±1.5e−45 до ±3.4e38
double	64-битное вещественное число с точностью 15-16	От ±5.0e−324 до ±1.7e308
decimal	128-битное вещественное число с точностью 28-29 знаков	От ±1.0 × 10e−28 до ±7.9 × 10e28
char	16-битный символ Unicode	От U+0000 до U+ffff
bool	Хранит значения true или false	true или false

- 16-битные символы (с префиксом “U”) используются для отображения различных языков, использующихся в мире

Знаковые целые

- Представляют и положительные, и отрицательные числа

Представление символьных типов данных и типов с плавающей точкой

- Переменные типов с плавающей точкой всегда заканчиваются символом “F” или “f” и всегда должны быть заключены в одинарные кавычки

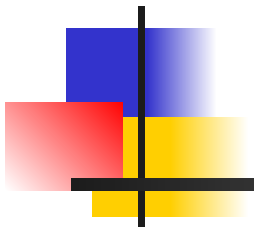
Классификация

Ссылочные типы данных хранят в памяти ссылки на объекты, содержащие значения.

Классификация ссылочных типов данных:

- Объект
  - Это базовый класс для всех предопределенных и пользовательских типов данных.
- Массив
  - Это определенная пользователем структура, содержащая значения одного типа данных, например, оценки студентов.
- Делегат
  - Это определенный пользователем тип, содержащий ссылку на один или более методов.
- Строка
  - Представляет строку символов Unicode.
  - Позволяет присваивать и изменять строковые значения.
  - Однажды созданная строка не может быть модифицирована.
- Интерфейс
  - Это тип определенного пользователем класса, который используется для сложного наследования.

Объект
Массив
Делегат
Строка
Интерфейс



Правила

Правила объявления переменных:

- Имя переменной может начинаться с буквы в верхнем или нижнем регистре. Имя может содержать буквы, цифры и символ подчеркивания (_).
- Первый символ в имени переменой должен быть буквой и не может быть цифрой. Подчеркивание также допустимый первый символ, но он не рекомендуется для начала имени.
- Язык C# чувствителен к регистру; таким образом переменные с именами count и Count - две разные переменные.
- Ключевые слова C# не могут быть использованы в качестве имен переменных. Если вам необходимо использовать ключевые слова C#, используйте символ ‘@’ как префикс.

Проверка

Имя переменной	Допустимо/недопустимо
Employee	Допустимо
student	Допустимо
_Name	Допустимо
Emp_Name	Допустимо
@goto	Допустимо
static	Недопустимо, поскольку слово ключевое
4myclass	Недопустимо, поскольку имя переменной не может начинаться с цифры
Student&Faculty	Недопустимо, поскольку имя переменной не может содержать специальный символ &

Объявление

Пример **бъявление одной переменой**

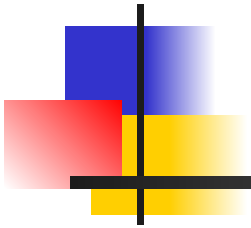
```
<data type> <variable name1> = <value1>, <variable name2>
= <value2>;
```

`data type`: допустимый тип данных.
`variable name`: допустимое имя переменной или идентификатор.
`value`: значение, присваиваемое переменной.

Синтаксис – Объявление нескольких переменных

```
<data type> <variable name1>, <variable
name2>,..,<variable nameN>;
```

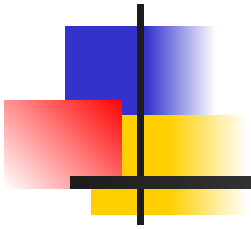
где,
`data type`: допустимый тип данных.
`variable name1, variable name2, variable nameN`: имена переменных или идентификаторы.



Занятие 2 - Комментарии и XML-документация

На втором занятии, **Комментарии и XML-документация**, вы научитесь:

- Описывать комментарии в C# приложениях.
- Объяснять XML-документацию исходного кода.



Определение 1-2

Комментарии

- Предоставляют информацию о части кода.
- Делают программу более читаемой.
- Помогают программисту объяснить цель использования редких переменных или методов.
- Идентифицируются с помощью специальных символов.
- Игнорируются компилятором при выполнении программы

- C# поддерживает три типа комментариев:
  - Однострочные комментарии
  - Многострочные комментарии
  - XML-комментарии

Определение 2-2

- Многострочные комментарии
  - Начинаются символами косой черты и звездочки (/*) и заканчиваются звездочкой, за которой следует косая черта (*/)
- В отличие от одно- и многострочных комментариев, XML-комментарии должны заключаться в XML-тэги. Вам необходимо создать XML-тэг для вставки XML-комментариев
  - XML-тэг должен быть вставлен между начальным и конечным

```
/// <summary>
/// Вы в тэге XML называемом summary.
/// </summary>
```

```
   перемножает два числа,
   делит результат на 2 и
   выводит частное */

int doMult = 5 * 20;
int doDiv = doMult / 2;
Console.WriteLine("Quotient is:" + doDiv)
```

Документация XML

```
class XMLComments
{
    /// <summary>
    /// This is the XML comment and can be extracted to a XML file.
    /// </summary>
    static void Main(string[] args)
    {
        Console.WriteLine("This program illustrates XML Comments");
    }
}
```

ко

ованные в коде. В этом случае вы можете создать

умент, который будет содержать всю необходимую

Синтаксис

```
csc /doc: <XMLfilename.xml> <CSharpfilename.cs>
```

где,

`XMLfilename.xml`: имя создаваемого XML файла .

`CSharpfilename.cs`: имя C# файла, из которого извлекаются XML-комментарии.

Предопределенные XML тэги

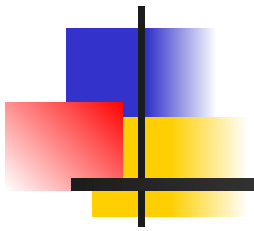
- XML-комментарии вставляются в XML-тэги.
- Эти тэги могут быть предопределенными или определенными пользователем.

Предопределенные теги	Описание
<c>	Устанавливает для текста шрифт кода
<code>	Устанавливает одну или более строк исходного кода или вывода программы.
<example>	Обозначает пример.
<param>	Описывает параметр метода или конструктора.
<returns>	Описывает возвращаемое методом значение.
<summary>	Резюмирует общую информацию о коде.

Занятие 3 - Константы и литералы

На третьем занятии, **Константы и литералы**, вы научитесь:

- Описывать константы в C#.
- Перечислять различные типы литералов.



Необходимость констант

- Константы в C# - это фиксированные значения, присвоенные идентификаторам, которые не меняются при выполнении кода.

- Константы определяются тогда, когда значения должны быть фиксированными и повторно используемыми или для предотвращения их модификации.

Константы

Синтаксис

- Константы могут быть определены для всех типов данных.

```
const float _pi = 3.14F;
float radius = 5;
float area = _pi * radius * radius;
Console.WriteLine("Area of the circle is ” + area);
```

объявляется как константа.

`data type`: Тип данных константы.

`identifier name`: Имя идентификатора, который содержит константу.

`value`: Фиксированное значение, не изменяющееся во время выполнения кода.

- Для объявления идентификатора как константы, используется ключевое слово `const` в объявлении идентификатора. Компилятор идентифицирует константы при компиляции по ключевому слову `const`.

Использование литералов 1-4

- В C# литерал – это значение, присвоенное переменной или константе.
- Логические литералы
- Целые литералы
- Вещественные литералы
- Символьные литералы
- Строковые литералы
- Null Literal

Литералы могут быть определены для любого типа данных C#.

Числовые литералы могут содержать суффикс в виде символа алфавита, определяющего тип данных литерала. Литералы могут быть в верхнем или нижнем регистре. Например, `string bookName = “Csharp”`.

Использование литералов 2-4

Целочисленные литералы

- Целочисленные литералы могут принимать два значения: int, long или ulong. Суффиксы для целых литералов включают U, L, UL или LU. U обозначает uint или ulong, L обозначает long. UL и

Пример

```
bool val = true;
```

Пример

логический литерал, присвоенный переменной val.

```
long val = 53L;
```

где,

`53L`: целый литерал, присвоенный переменной val.

Использование литералов 3-4

Символьные литералы

- Символьные литералы присваиваются типам данных `char`. Символьный литерал всегда заключается в одинарные кавычки.

Пример

```
char val = 'A';
```

где,

`A`: символьный литерал, присвоенный переменной `val`.

Вещественные литералы

- Вещественные литералы принадлежат типам данных `float`, `double` и `decimal`. На конце указывается суффикс, добавляемый после присваиваемого значения. Вещественный литерал может заканчиваться на F, D или M. F обозначает `float`, D обозначает `double`, а M обозначает `decimal`.

```
float val = 1.66F;
```

где,

`1.66F`: вещественный литерал, присвоенный переменной val.

Использование литералов 4-4

Строковые литералы

- Есть два типа строковых литералов в C#, обычные и дословные.
- [illegible] строковый литерал - это стандартная строка.
- [illegible] строковый литерал, но имеет префикс @.
- [illegible]

Null Literal

- Null-литерал имеет только одно значение - `null`.

Пример

```
string email = null;
```

где,

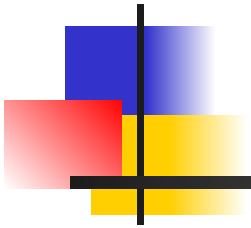
`null`: Определяет email, не ссылающийся ни на один объект (ссылку).

Пример

```
string mailDomain = "@gmail.com";
```

где,

`@gmail.com`: дословный строковый литерал.



Занятие 4 - Ключевые слова и escape-последовательности

На четвертом занятии, **Ключевые слова и escape-последовательности**, вы научитесь:

- перечислять ключевые слова в C#.
- Перечислять и объяснять символы escape-последовательностей.

Ключевые слова

- Ключевые слова - это зарезервированные слова, которые обрабатываются компилятором отдельно.

abstarct	bool	break	byte	case	catch
char	class	const	continue	default	double
enum	else	false	finally	float	for
foreach	goto	if	int	interface	long
namespace	new	public	private	protected	return
sbyte	short	static	string	struct	switch
throw	true	try	ushort	void	while

- Ключевые слова не могут быть использованы в именах переменных, методов или классов (за исключением случаев использования префикса в виде символа “@”).

Необходимость в символах escape-последовательностей

- Рассмотрим фонд заработной платы в какой-либо организации.
- Одна из его функций - показывать месячную зарплату, при этом каждое значение должно выводиться в новой строке.
- Программист хочет написать код, который всегда печатает зарплату в новой строке вне зависимости от длины строки, представляющей размер заработной платы.
- Для этого используются escape-последовательности.

Определение

- Символ escape-последовательности - это символ обратного слэша сообщает компилятору, специальный символ, следующий за символом что следующий символ отмечен как непечатный. обратного слэша (\).
- Например, \n используется для добавления новой
- Символы escape-последовательностей строки подобно клавише `Enter` на клавиатуре. используются для реализации специальных непечатных символов, например, новой строки,
- В C# символы escape-последовательности должны одиночного пробела или возврата каретки. всегда заключаться в двойные кавычки.
- Непечатные символы используются при форматированном выводе для повышения удобочитаемости.

Символы escape-последовательностей в C#

В C# существует много символов escape-последовательностей, которые используются для

Пример

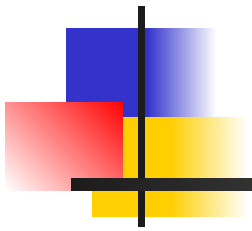
```
string str = "\u0048\u0065\u006C\u006C\u006F";
Console.Write("\t" + str + "!\n");
Console.WriteLine("David\u0020\"2007\" ");
```

Вывод

```
 Hello!
David "2007"
```

\	Возврат каретки.
\t	Горизонтальная табуляция.
\	Одинарная кавычка, необходима для символьных литералов
\	
\	
\	
\	
\0	Символ NULL.
\a	Предупреждение.
\b	Backspace.
\f	Перевод страницы.
\n	Новая строка.

шестнадцатеричное представление (две цифры). Например, \x61 представляет символ 'a'.



Занятие 5 - Ввод и вывод

На последнем занятии, **Ввод и вывод**, вы научитесь:

- Описывать методы консольного вывода в C#.
- Описывать операции консольного ввода в C#.
- Объяснять спецификаторы форматирования в C#.
- Объяснять спецификаторы форматирования даты и времени.

Консольные операции

- Консольные операции – это задачи, выполняемые в интерфейсе командной строки с использованием исполняемых команд.
- Консольные приложения выполняются в командной строке.
- Все консольные приложения состоят из трех потоков, представляющих собой последовательности байт.
- Эти потоки связаны с устройствами ввода и вывода компьютерной системы и обрабатывают операции ввода и вывода.

Стандартный ввод

Стандартный поток in берет ввод и передает его в консольное приложение для обработки.

Стандартный вывод

Стандартный поток out показывает вывод на мониторе.

Стандартный err

Стандартный поток err показывает сообщения об ошибках на мониторе.

- Консольные операции используются в программировании, потому что они легко контролируются операционной системой потому что они зависят от устройств ввода и вывода компьютерной системы.

Методы вывода 1-2

- В C# все консольные операции содержатся в классе `Console` пространства имен `System`.
- Записывает любой тип данных.

Console.Write()

Синтаксис вывода данных в консоль вам необходим

```
Console.Write("<data>" + variables);
```

где,

`data`: Заданная строка или символы escape-последовательности, заключенные в двойные кавычки.

`variables`: Заданные имена переменных, значения которых должны быть выведены на консоль.

- `Console.Write()`
- `Console.WriteLine()`

Методы вывода 2-2

Пример

- **iteLine()**
- Выводит любой тип данных и символ конца строки

```
Console.WriteLine("C# is a powerful programming
language");
Console.WriteLine("C# is a powerful");
Console.WriteLine("programming language");
Console.Write("C# is a powerful");
Console.WriteLine(" programming language");
```

В коде показана разница между двумя методами.

Вывод

```
.WriteLine("<data>" + variables);
```

```
C# is a powerful programming language
C# is a powerful
programming language
C# is a powerful programming language
```

Указатели места заполнения

Пример

- ы `WriteLine()` и `Write()` принимают список параметров для форматирования текста перед выводом основании номеров переданных в списке.

```
int number, result;
number = 5;
result = 100 * number;
Console.WriteLine("Result is {0} when 100 is multiplied
by {1}", result,number);
result = 150 / number;
Console.WriteLine("Result is {0} when 150 is divided by
{1}", +result, number);
```

Вывод

- Например, для указания позиции первого параметра вы

```
Result is 500 when 100 is multiplied by 5
Result is 30 when 150 is divided by 5
```

Методы ввода

Пример

- Для чтения данных вам необходим стандартный поток ввода

```
string name;
Console.Write("Enter your name: ");
name = Console.ReadLine();
Console.WriteLine("You are {0}",name);
```

Вывод

```
Enter your name: David Blake
You are David Blake
```

- Методы ввода, которые позволяют программному стандартного устройства ввода:
  - **`Console.Read()`** - Читает один символ.
  - **`Console.ReadLine()`** - Читает строку.

Методы преобразования

Вывод ReadLine() может также использоваться для ввода целых значений

```
Enter your name: David Blake
Enter your age: 34
Enter the salary: 3450.50
Name: David Blake, Age: 34, Salary: 3450.50
```

```
userName = Console.ReadLine();
Console.Write("Enter your age: ");
age = Convert.ToInt32(Console.ReadLine());
Console.Write("Enter the salary: ");
salary = Convert.ToDouble(Console.ReadLine());
Console.WriteLine("Name: {0}, Age: {1}, Salary: {2} ",
userName, age, salary);
```

Определение спецификаторов числового форматирования

Синтаксис

```
Console.WriteLine("{format specifier}", <variable name>);
```

где,

`format specifier`: числовой спецификатор форматирования.
`variable name`: имя целой переменной.

- В C# вы можете преобразовывать числовые значения в различные форматы.
- Это используется в методах вывода класса `Console`.
- Например, вы можете выводить большие числа в экспоненциальном формате.

Некоторые спецификаторы числового форматирования

Пример

```
int num = 456;
Console.WriteLine("{0:C}", num);
Console.WriteLine("{0:D}", num);
Console.WriteLine("{0:E}", num);
```

Вывод

```
$456.00
456
4.560000E+002
```

Спецификатор форматирования	Название	Описание
C или c		
D или d		
E или e	(Экспоненциальный)	

- Например, если вы используете спецификатор вида C3, три нуля будут вставлены после decimal location для данного числа.

Еще числовые спецификаторы форматирования

Спецификатор форматирования	Название	Описание
F		символа минус.
N	Число	Числа преобразовываются в форму “-d.ddd.ddd.ddd…” где каждая ‘d’
X или x	Шестнадцатеричное	Число преобразуется в строку шестнадцатеричных цифр. Используйте “X” для “ABCDEF” и “x” для “abcdef”.

Пример

```
int num = 456;
Console.WriteLine(“{0:F}”, num);
Console.WriteLine(“{0:N}”, num);
Console.WriteLine(“{0:X}”, num);
```

Вывод

```
456.00
456.00
1C8
```

Стандартные спецификаторы форматирования даты и времени

Синтаксис

```
Console.WriteLine("{format specifier}", <datetime object>);
```

где,

`format specifier`: спецификатор форматирования даты и времени.
`datetime object`: объект класса `DateTime`.

- Спецификаторы форматирования даты и времени - это специальные символы, позволяющие вам форматах.

Гринвичское время (GMT), то вы можете выводить его с использованием аббревиатуры GMT, используя спецификаторы форматирования даты и времени.

- Спецификаторы форматирования даты и времени позволяют вам выводить дату и время в 12-ти и 24-х часовых форматах.

Некоторые стандартные спецификаторы форматирования даты и времени 1-2

Спецификатор форматирования	Название	Описание
F	Полностью дата/время (длинное время)	Выводит дату в соответствии с длинным шаблоном. Формат по умолчанию “dddd*, MMMM* dd, yyyy HH*:mm*:ss*”
D	Длинная дата	Выводит дату в соответствии с длинным шаблоном. Формат по умолчанию “dddd*, MMMM*, dd, yyyy”
g	Обычное дата/время (короткое время)	Выводит дату в коротком формате и время коротком формате, разделены пробелом. Формат по умолчанию “MM/dd/yyyy HH*:mm*”
f	Полностью дата/время (короткое время)	Выводит дату в длинном формате (разделенные пробелом). Формат по умолчанию “dddd*, MMMM* dd, yyyy HH*:mm*”.

Некоторые стандартные спецификаторы форматирования даты и времени 2-2

Вывод

```
Short date format(d): 23/04/2007
Long date format (D): Monday, April 23, 2007
Full date with time without seconds (f):Monday, April
23, 2007 12:58 PM
Full date with time with seconds (F):Monday, April 23,
2007 12:58:43 PM
Short date and short time without seconds
(g):23/04/2007 12:58 PM
```

```
// Возвращает полностью дату и время с секундами
Console.WriteLine("Full date with time with seconds
(F):{0:F}", dt);
// Возвращает короткую дату и время без секунд
Console.WriteLine("Short date and short time without
seconds (g):{0:g}", dt);
```

Еще стандартные спецификаторы форматирования даты и времени 1-2

Спецификатор форматирования	Название	Описание
G	Длинное время	Выводит дату в коротком формате и время в длинном формате (разделенные пробелом). Формат по умолчанию “M/dd/yyyy HH*:mm*:ss*”.
y или Y	Шаблон месяцев	Выводит только месяц и год. Формат по умолчанию “YYYY MMMM*”.
m или M	День месяца	Выводит только месяц и день. Формат по умолчанию “MMMM* dd”.
T	Короткое время	Выводит время в соответствии с коротким шаблоном. Формат по умолчанию “HH*: mm*”.

Еще стандартные спецификаторы форматирования даты и времени 2-2

Вывод

```
Short date and short time with seconds (G):23/04/2007
12:58:43 PM
Month and day (m):April 23
Short time (t):12:58 PM
Short time with seconds (T):12:58:43 PM
Year and Month (y):April, 2007
```

```
Console.WriteLine("Month and day (m):{0:m}", dt);
// Возвращает короткое время
Console.WriteLine("Short time (t):{0:t}", dt);
// Возвращает короткое время с секундами
Console.WriteLine("Short time with seconds (T):{0:T}", dt);
// Возвращает год и месяц - Y также может быть использовано
Console.WriteLine("Year and Month (y):{0:y}", dt);
```

Заключение 1-3

Переменные и типы данных в C#

- Переменные - это именованные области компьютерной памяти и хранилища данных.
- Тип данных указывает вид данных, хранящихся в переменной.
- Тип данных может быть любым типом-значением или ссылочным типом.

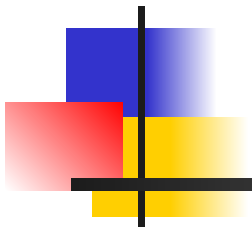
Комментарии и XML-документация

- Комментарии используются для предоставления детальных пояснений о различных сферах кода.
- Вы можете вставить комментарий, добавив двойной обратный слэш (//) перед пояснением к коду.
- XML-комментарии начинаются с тройного обратного слэша (///) и заключаются в XML-тэги.
- Вы можете создать XML-документ, который будет содержать все XML-комментарии.

Заключение 2-3

Ключевые слова и Escape последовательности

- Ключевые слова – это специальные слова, которые вы не можете использовать в C#.
- Ключевые слова – это специальные слова, которые вы предопределяете в C#.
- Ключевые слова – это специальные слова, которые вы не можете использовать в C#.
- Ключевые слова – это специальные слова, которые вы предопределяете в C#.
- Литералы – это фиксированные значения, которые вы не можете изменить при выполнении программы.
- Литералы – это фиксированные значения, которые вы не можете изменить при выполнении программы.
- Литералы – это фиксированные значения, которые вы не можете изменить при выполнении программы.
- Литералы – это фиксированные значения, которые вы не можете изменить при выполнении программы.
- Escape-последовательности – это специальные символы, которые для компилятора имеют специальное значение.
- Escape-последовательности – это специальные символы, которые для компилятора имеют специальное значение.
- Escape-последовательности – это специальные символы, которые для компилятора имеют специальное значение.
- Escape-последовательности – это специальные символы, которые для компилятора имеют специальное значение.
- Символы escape-последовательностей позволяют выводить непечатные символы.



Заключение 3-3

Ввод и вывод

- `Console.Read()` и `Console.ReadLine()` - методы ввода, которые всегда возвращают строковое значение.
- Вы можете преобразовать строковое значение в другой тип данных, используя методы преобразования класса `Convert`.
- `Console.Write()` и `Console.WriteLine()` - методы вывода.
- Спецификаторы форматирования позволяют вам настроить вывод в окно консоли.