В прошлой теме был определен метод SayHello, который выводит на консоль некоторое сообщение:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | void SayHello()  {  Console.WriteLine("Hello");  }    SayHello(); // Hello |

Но минусом подобного метода является то, что он выводит одно и то же сообщение. И было бы неплохо, если бы мы могли бы динамически определять, какое сообщение будет выводить метод на экран, то есть передать из вне в метод это сообщение. Для этого в языке C# мы можем использовать параметры.

**Параметры** позволяют передать в метод некоторые входные данные. Параметры определяются через заятую в скобках после названия метода в виде:

|  |  |
| --- | --- |
| 1  2  3  4 | тип\_метода имя\_метода (тип\_параметра1 параметр1, тип\_параметра2 параметр2, ...)  {  // действия метода  } |

Определение параметра состоит из двух частей: сначала идет тип параметра и затем его имя.

Например, определим метод PrintMessage, который получает извне выводимое сообщение:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | void PrintMessage(string message)  {  Console.WriteLine(message);  }    PrintMessage("Hello work"); // Hello work  PrintMessage("Hello METANIT.COM"); // Hello METANIT.COM  PrintMessage("Hello C#"); // Hello C# |

Здесь метод PrintMessage() принимает один параметр, который называется message и имеет тип string.

Чтобы выполнить метод, который имеет параметры, при вызове после имени метода в скобках ему передаются значения для его параметров, например:

|  |  |
| --- | --- |
| 1 | PrintMessage("Hello work"); |

Здесь параметру message передается строка "Hello work". Значения, которые передаются параметрам, еще называются **аргументами**. То есть передаваемая строка "Hello work" в данном случае является аргументом.

Иногда можно встретить такие определения как **формальные параметры** и **фактические параметры**. Формальные параметры - это собственно параметры метода (в данном случае message), а фактические параметры - значения, которые передаются формальным параметрам. То есть фактические параметры - это и есть аргументы метода.

Определим еще один метод, который складывает два числа:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | void Sum(int x, int y)  {  int result = x + y;  Console.WriteLine($"{x} + {y} = {result}");  }    Sum(10, 15); // 10 + 15 = 25 |

Метод Sum имеет два параметра: x и y. Оба параметра представляют тип int. Поэтому при вызове данного метода нам обязательно надо передать на место этих параметров два числа. Внутри метода вычисляется сумма переданных чисел и выводится на консоль.

При вызове метода Sum значения передаются параметрам по позиции. Например, в вызове Sum(10, 15) число 10 передается параметру x, а число 15 - параметру y.

Также параметры могут использоваться в сокращеной версии метода:

|  |  |
| --- | --- |
| 1  2  3 | void Sum(int x, int y) => Console.WriteLine($"{x} + {y} = { x + y }");    Sum(10, 15); // 10 + 15 = 25 |

Передаваемые параметру значения могут представлять значения переменных или результат работы сложных выражений, которые возвращают некоторое значение:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | void Sum(int x, int y) => Console.WriteLine($"{x} + {y} = { x + y }");    int a = 10, b = 15, c = 6;  Sum(a, b); // 10 + 15 = 25  Sum(3, c); // 3 + 6 = 9  Sum(14, 4 + c); // 14 + 10 = 24 |

Если параметрами метода передаются значения переменных, то таким переменным должно быть присвоено значение. Например, следующая программа не скомпилируется:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | void Sum(int x, int y)  {  Console.WriteLine($"{x} + {y} = { x + y }");  }    int a;  int b = 15;  Sum(a, b); // ! Ошибка |

### Соответствие параметов и аргументов по типу данных

При передаче значений параметрам важно учитывать тип параметров: между аргументами и параметрами должно быть соответствие по типу. Например:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | void PrintPerson(string name, int age)  {  Console.WriteLine($"Name: {name} Age: {age}");  }    PrintPerson("Tom", 24); // Name: Tom Age: 24 |

В данном случае первый параметр метода PrintPerson() представляет тип string, поэтому при вызове метода мы должны передать этому параметру значение типа string, то есть строку. Второй параметр представляет тип int, поэтому должны передать ему целое число, которое соответствует типу int.

|  |  |
| --- | --- |
| 1 | PrintPerson("Tom", 24); |

Также мы можем передать параметрам значения тех типов, которые автоматически могут быть преобразованы в тип параметров. Например:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | void PrintPerson(string name, int age)  {  Console.WriteLine($"Name: {name} Age: {age}");  }    byte b = 37;  PrintPerson("Tom", b); // Name: Tom Age: 37 |

Здесь параметру типа int передается значение типа byte, но комилятор может автоматически преобразовать значение типа byte к тиу int. Поэтому здесь ошибки не возникнет. Какие преобразования типов могут быть выполнены автоматически, рассматривалось в одной из предыдущих тем: [Преобразования базовых типов данных](https://metanit.com/sharp/tutorial/2.2.php)

Данные других типов мы передать параметров не можем. Например, следующий вызов метода PrintPerson будет ошибочным:

|  |  |
| --- | --- |
| 1 | PrintPerson(45, "Bob"); // Ошибка! несоответствие значений типам параметров |

### Необязательные параметры

По умолчанию при вызове метода необходимо предоставить значения для всех его параметров. Но C# также позволяет использовать необязательные параметры. Для таких параметров нам необходимо объявить значение по умолчанию. Также следует учитывать, что после необязательных параметров все последующие параметры также должны быть необязательными:

|  |  |
| --- | --- |
| 1  2  3  4 | void PrintPerson(string name, int age = 1, string company = "Undefined")  {  Console.WriteLine($"Name: {name} Age: {age} Company: {company}");  } |

Здесь параметры age и company являются необязательными, так как им присвоены значения. Поэтому при вызове метода мы можем не передавать для них данные:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | void PrintPerson(string name, int age = 1, string company = "Undefined")  {  Console.WriteLine($"Name: {name} Age: {age} Company: {company}");  }    PrintPerson("Tom", 37, "Microsoft"); // Name: Tom Age: 37 Company: Microsoft  PrintPerson("Tom", 37); // Name: Tom Age: 37 Company: Undefined  PrintPerson("Tom"); // Name: Tom Age: 1 Company: Undefined |

Консольный вывод программы:

Name: Tom Age: 37 Company: Microsoft  
Name: Tom Age: 37 Company: Undefined  
Name: Tom Age: 1 Company: Undefined

### Именованные параметры

В предыдущих примерах при вызове методов значения для параметров передавались в порядке объявления этих параметров в методе. То есть аргументы передавались параметрам **по позиции**. Но мы можем нарушить подобный порядок, используя именованные параметры:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | void PrintPerson(string name, int age = 1, string company = "Undefined")  {  Console.WriteLine($"Name: {name} Age: {age} Company: {company}");  }    PrintPerson("Tom", company:"Microsoft", age: 37); // Name: Tom Age: 37 Company: Microsoft  PrintPerson(age:41, name: "Bob"); // Name: Bob Age: 41 Company: Undefined  PrintPerson(company:"Google", name:"Sam"); // Name: Sam Age: 1 Company: Google |

Для передачи значений параметрам о имени при вызове метода указывается имя параметра и через двоеточие его значение: name:"Tom"

Консольный вывод программы:

Name: Tom Age: 37 Company: Microsoft  
Name: Bob Age: 41 Company: Undefined  
Name: Sam Age: 1 Company: Google