

Лекция 2

управляющие конструкции, циклы и
функции

Блок кода

- Блок - часть кода заключенная в фигурные скобки
 - например: `{ int i = 0; }`
- Блоки могут быть вложенными:
 - `{ int i = 0;`
 `{ int j = 1; }`
 `}`
- Блоки могут быть самостоятельными (как в примерах выше), а могут относиться к `if`, `else`, `for`, `while` и тд. Функция - это тоже пример блока кода.

Управление ходом выполнения программы

- Условия
 - выполнить блок кода если условие верно
- Выбор варианта
 - выполнить блок кода соответствующий заданному значению
- Циклы
 - повторить блок кода заданное количество раз

Условия

```
if ( condition ) {  
    Console.WriteLine(“Условие истинно”);  
}  
else {  
    Console.WriteLine(“Условие ложно”);  
}
```

Что может находиться внутри if (*):

- переменные типа **bool (true, false)**
- функции возвращающие **bool**
- Числа (**0 - ложь, любое другое - истина**)
- Операции сравнения (**<, >, !=** и д.р.)
- Группы логических выражений (**||, &&**)

```
Console.WriteLine ("Представьтесь:");  
  
string name = Console.ReadLine ();  
  
if (name.Length == 0) {  
    name = "Аноним";  
}  
  
Console.WriteLine ("Привет, {0}!", name);
```

```
Random rand = new Random ();  
int a = rand.Next (100);  
int b = rand.Next (100);  
int c = rand.Next (100);  
Console.WriteLine ("[{0}, {1}, {2}]", a, b, c);  
if (a >= b && a >= c) {  
    Console.WriteLine ("max = " + a);  
} else if (b >= a && b >= c) {  
    Console.WriteLine ("max = " + b);  
} else {  
    Console.WriteLine ("max = " + c);  
}
```

Оператор выбора

```
switch (variable){  
  case CONST1:  
    <some code>  
    break;  
  case CONST2:  
    <some other code>  
    break;  
  ...  
  default:  
    <default code block>  
    break;  
}
```

variable type:

int, char, enum,
string, bool(?)

Но не:

float, double, object


```
Console.WriteLine (@"
```

```
1. Расчет первого выражения
```

```
2. Расчет второго выражения
```

```
3. Выход
```

```
");
```

```
int choice = int.Parse(Console.ReadLine ());
```

```
switch (choice) {
```

```
case 1:
```

```
    Console.WriteLine ("2*2 = " + (2*2));
```

```
    break;
```

```
case 2:
```

```
    Console.WriteLine ("2+2 = " + (2+2));
```

```
    break;
```

```
case 3:
```

```
    return;
```

```
}
```

```
string name = Console.ReadLine ();
```

```
bool admin;
```

```
switch (name) {
```

```
case "Василий":
```

```
case "Петр":
```

```
    admin = true;
```

```
    break;
```

```
default:
```

```
    admin = false;
```

```
    break;
```

```
};
```

```
double x = double.Parse (Console.ReadLine());
```

```
switch (x) {
```

```
case 0.1:
```

```
    Console.WriteLine ("x = 0.1");
```

```
    break;
```

```
case 0.2:
```

```
    Console.WriteLine ("x = 0.2");
```

```
    break;
```

```
default:
```

```
    Console.WriteLine ("x has some other value");
```

```
    break;
```

```
}
```

Нельзя использовать
double в switch

Циклы

- Повторить определенное количество раз
`for (start; condition; increment)`
`foreach(var in container)` - о нем позже
- Повторять пока выполняется условие
`do {} while(condition)`
- Пока выполняется условие - повторять
`while(condition)`

```
do {  
    Console.WriteLine ("Hello World!");  
} while (true);
```

```
while (true) {  
    Console.WriteLine ("Hello World!");  
}
```

```
do {  
    Console.WriteLine ("Hello World!");  
} while (false);
```

```
while (false) {  
    Console.WriteLine ("Hello World!");  
}
```

```
Random rand = new Random ();  
int i;  
do {  
    i = rand.Next (-100, 10);  
    Console.WriteLine ("i = " + i);  
} while(i < 0);
```

```
for (int i = 0; i < 10; i++) {  
    Console.WriteLine ("Hello World!");  
}
```

for

```
int j = 0;  
while ( j < 10 ) {  
    j++;  
    Console.WriteLine ("Hello World!");  
}
```

while


```
string hello = "Hello World!";  
for (int i = 0; i < hello.Length; i++) {  
    Console.Write (hello [i]);  
}
```

Досрочный выход из цикла

```
Random rand = new Random ();
```

```
int number;
```

```
for (int i = 0; i < 10; i++) {  
    number = rand.Next (-10, 10);  
    if (number > 0) {  
        break;  
    }  
}
```

Досрочный переход к следующей итерации

```
for (item = 1; item < Total_items; item++) {  
    //обработываем элемент  
    if (Закончили_обработку_текущего_элемента)  
        continue;  
    //дополнительная обработка  
}
```

Функции, они же методы

- Помогают разбить код на логические части
- Позволяют использовать код повторно

```
[модификаторы] <возвращаемый тип>  
<имя> ([параметры]){  
    [тело функции]  
}
```

```
static void hello(){  
    Console.WriteLine("Hello!");  
}
```

```
public static void Main (string[] args)  
{  
    hello ();  
    hello ();  
    hello ();  
}
```

Параметры

```
static void printValue(int val){  
    Console.WriteLine(val);  
}  
  
public static void Main (string[] args)  
{  
    printValue (112);  
    printValue (911);  
}
```

Возвращаемое значение

```
static int printAndInc(int i){  
    Console.WriteLine(i);  
    return i + 1;  
}  
  
public static void Main (string[] args)  
{  
    printAndInc(  
        printAndInc(  
            printAndInc (1)  
        ));  
}
```



```
static double readValue(  
    string prompt,  
    double minValue,  
    double maxValue)  
{  
    double result;  
    do {  
        Console.WriteLine(prompt);  
        result = double.Parse(Console.ReadLine());  
    } while (result < minValue || result > maxValue);  
    return result;  
}
```

```
public static void Main (string[] args)
{
    double age = readValue
        ("Введите возраст: ", 0, 120);
    double height = readValue
        ("Введите рост: ", 1.0, 2.30);
}
```

Сигнатура метода

Название метода + список параметров
определяют уникальную сигнатуру метода

В классе не может быть двух методов с
одинаковой сигнатурой!

Передача параметров по значению

- Если мы не указали иного, параметры ВСЕГДА передаются **по значению**
- Функция не может повлиять на значение переменной переданной в эту функцию

```
static int printAndInc(int i){  
    i = i + 1;  
    Console.WriteLine ("i is " + i);  
    return i;  
}
```

```
public static void Main (string[] args)  
{  
    int i = 10;  
    printAndInc (i);  
    Console.WriteLine("i is " + i);  
}
```

Передача параметров по ссылке

- ключевое слово **ref** и в описании функции и при ее вызове
- передается не значение переменной, а ее адрес в памяти
- функция **МОЖЕТ** изменить значение внешней переменной

```
static int printAndInc(ref int i){  
    i++;  
    Console.WriteLine ("i is " + i);  
    return i;  
}  
  
public static void Main (string[] args)  
{  
    int test = 10;  
    printAndInc (ref test);  
    Console.WriteLine("test is " + test);  
}
```

Выходные параметры

- ключевое слово **out** должно указываться и **при объявлении** функции и **при ее вызове**
- функция **не может** прочесть значение выходного параметра, а только записать
- функция **обязана** записать значение выходного параметра


```
static void readPerson ( out string name, out int age )  
{  
    name = readString ( "Enter your name : " ) ;  
    age = readInt ( "Enter your age : ", 0, 100 ) ;  
}
```

```
public static void Main (string[] args)  
{  
    string name;  
    int age;  
    readPerson (out name, out age);  
}
```

Область видимости (scope)

Область видимости - блок и вложенные в него блоки в котором определена переменная

- Переменная объявленная внутри блока может использоваться только в нем и вложенных блоках
- Переменная не видна за пределами своего блока
- Внутри одной области видимости может быть объявлена только одна переменная с одим именем
- В разных областях видимости могут объявляться переменные с одинаковым именем. Но это будут разные переменные, с разными данными внутри

```
{  
    int i = 0;  
    {  
        //ошибка, в этой области видимости уже  
        //определена переменная i  
        int i = 2;  
    }  
}
```

```
{  
    {  
        int i = 0;  
    }  
    {  
        //Нет ошибки, т.к. переменные в разных областях  
        //видимости  
        int i = 2;  
    }  
}
```

Область видимости переменных объявленных в for(;;) - это блок идущий сразу за for()

```
for(int j = 0; j < 10; j++){
```

```
    //double i = 0.1;
```

```
}
```

```
{
```

```
    int j = 10; //норма, другая область
```

```
}
```

```
int j = 30; //ошибка, переменная j уже объявлена в  
           //дочерних блоках
```

Итого

- переменные видны только в пределах своего блока
- в пределах одного блока не может быть двух переменных с одним именем
- в разных (не вложенных) блоках переменные с одним именем - разные переменные с разным содержимым