Кубанский государственный
аграрный университет

Факультет прикладной
информатики

# Лекция 2
## СИНТАКСИС И ПРОГРАММНЫЕ КОНСТРУКЦИИ VBA

1. Типы данных.
2. Переменные и константы.
3. Массивы.
4. Операторы.
5. Циклы.
6. Процедуры и функции.

Кубанский государственный аграрный университет

Факультет прикладной информатики

# Синтаксис VBA

- VBA нечувствителен к регистру;
- комментарий одинарная кавычка ( ') или команда REM;
- символьные значения должны заключаться в двойные кавычки;
- максимальная длина любого имени в VBA (переменные, константы, процедуры) — 255 символов;

- начало нового оператора — перевод на новую строку;
- ограничений на максимальную длину строки нет;
- несколько операторов в одной строке разделяются двоеточиями: MsgBox «Проверка 1» : MsgBox «Проверка 2»
- для удобства чтения можно объединить несколько физических строк в одну логическую при помощи пробела: MsgBox "Сообщение пользователю" _ & vUserName .

| Тип данных | Описание и диапазон значения |
|---|---|
| Byte | Хранение положительных чисел от 0 до 255 |
| Boolean | Хранение логических значений: True, False |
| Currency | Хранение чисел для точных вычислений в диапазоне<br>от -922337203685477,5808 до 922337203685477,5807 |
| Date | Хранение даты и времени.<br>Даты от 1.01.100 до 31.12.9999<br>Время от 00:00:00 до 23:59:59 |
| Double | Хранение чисел двойной точности<br>от $-1,79769313486232*10^{308}$ до $-4,94065645841247*10^{-324}$<br>от $4,94065645841247*10^{-324}$ до $1,79769313486232*10^{308}$ |
| Integer | Хранение целых чисел от -32768 до 32767 |
| Long | Хранение целых чисел от -2147483648 до 2147483647 |
| Single | Хранение чисел одинарной точности<br>от $-3,402823*10^{38}$ до $-1,401298*10^{-45}$<br>от $1,401298*10^{-45}$ до $3,402823*10^{38}$ |
| String | Хранение текста длиной до 2 млрд. символов |
| Variant | Хранение любого типа данных |

# Тип Date

*VBA использует тип Date для хранения дат и времени.*

При работе с этим типом данных следует иметь ввиду, что VBA-типы Date не являются такими же типами, как в рабочих листах Excel, хотя во многом и схожи с ними.

*Например, базовой датой для VBA-типа Date является 30 декабря 1899 года, а в Excel - 1 января 1900 года.*

*VBA-тип Date является типом последовательных дат*. VBA использует отрицательные числа для представления дат ранее базовой даты (30 декабря 1899), и положительные - для дат после базовой. Число 0 представляет саму дату 30.12.1899.

В значении последовательной даты целая часть - это общее число дней от базовой даты. Дробная часть (цифра справа от десятичного знака) - эти цифры обозначают время дня как часть дня. Один час - это 1/24 дня, одна минута - 1/1440 дня, секунда - 1/86400 дня.

Можно вычитать одну дату из другой, добавлять к дате или вычитать числа для изменения ее значения. В VBA имеется несколько встроенных процедур для отдельного извлечения года, месяца, дня, часов, минут и секунд переменной типа Date.

# Числа

VBA имеет шесть различных численных типов данных: *Byte, Integer, Long, Single, Double, Currency*.

Они используются для хранения чисел в различных форматах, в зависимости от конкретного типа.

# Целые числа

*Integer* - это целое число без дробной части (целые числа никогда не содержат десятичного знака, даже если десятичная часть равна нулю).

VBA предоставляет три типа целых данных: *Byte, Integer, Long*.

*Byte, Integer, Long* типы требуют меньше памяти для хранения чисел, чем другие численные типы данных VBA, а математические операции и операции сравнения над числами указанных типов быстрее, чем эти же операции для численных типов данных с плавающей точкой.

VBA автоматически преобразует данные типов Byte, Integer, Long в текст, когда они выводятся на экран, используя такие процедуры, как MsgBox.

# Числа с плавающей точкой

Числа с плавающей точкой могут иметь любое число цифр до или после десятичной точки (в пределах границ конкретного типа данных). Иногда их называют *действительными числами*. Этот тип данных используют тогда, когда требуется хранить числа, имеющие дробную часть.

VBA имеет два типа данных с плавающей точкой: *Single, Double*.

Числа, хранимые с использованием типа Single, называются числами одинарной точности.

Числа, хранимые с использованием типа Double, называются числами двойной точности.

Следует иметь ввиду, что операции, выполняемые над числами с плавающей точкой, немного медленнее подобных операций над другими численными типами данных. Кроме того, числа, хранимые как типы данных с плавающей точкой, могут быть подвержены ошибкам округления.

Если число с плавающей точкой очень большое или очень малое, VBA отображает его в экспоненциальном представлении.

# Тип данных Currency

Это число с фиксированной точкой, т.е., десятичная точка всегда находится в одном и том же месте - справа от точки всегда имеются четыре цифры. Этот тип данных используется при денежных вычислениях, когда требуется высокая точность.

# Текстовые строки

Любые текстовые данные, сохраняемые в VBA, называются строками. Для их хранения предназначен тип данных *String*.

Строка может содержать текстовые символы любых типов: буквы, цифры, знаки пунктуации, разделительные символы и пр.

Строки всегда заключаются в двойные кавычки.

Существует две категории строк: *строки переменной длины* и *строки фиксированной длины.* По умолчанию в VBA все строки переменной длины.

# Логические значения

VBA-программа принимает то или иное "решение", основываясь на различных условиях. Любое условие может принимать два значения: *True* (Истина) или *False* (Ложь). Логические значения True и False называются булевскими значениями, а тип данных - *Boolean*.

Данные типа **Variant** принимают характеристики определенного типа, который они сохраняют в данный момент. Например, если данные типа Variant содержат строковые данные, Variant принимает характеристики типа String. VBA использует для данных типа Variant наиболее компактное представление, возможное для конкретных значений, находящихся в данных.

Можно еще использовать пользовательские типы данных, но их вначале нужно определить при помощи выражения Type. Обычно пользовательские типы данных используются как дополнительное средство проверки вводимых пользователем значений (классический пример — почтовый индекс).

Самым простым способом создания переменной является использование ее в операторе VBA. VBA создает переменную и тут же резервирует ячейку памяти для данной переменной.

Сохранение значения данных в переменной называется присваиванием переменной. Присваивание выполняется с помощью оператора присваивания, представляемого знаком (=). Например, A = 145.

Создание переменной путем ее использования в операторе называется неявным объявлением переменной. Все переменные, которые VBA создает неявным объявлением переменной, имеют тип данных Variant.

VBA предоставляет возможность выполнять явное объявление переменных. Объявлять переменные явно лучше в начале программы, как это делается во всех языках программирования.

Явно объявить переменную можно как в начале блока, так и в том произвольном месте, где возникла необходимость использовать новую переменную. При объявлении переменной определяются ее тип и область видимости – область, где имя переменной видимо и, значит, возможен доступ к ее значению. Переменные можно объявлять на двух уровнях – уровне процедуры и уровне модуля.

Для объявления переменных используются операторы Dim, Public, Private и Static. Если нет никаких особых требований, то есть смысл всегда выбирать область видимости Dim.

**Dim** — используется в большинстве случаев. Если переменная объявлена как Dim в области объявлений модуля, она будет доступна во всем модуле, если в процедуре — только на время работы этой процедуры;

**Private** — при объявлении переменных в VBA значит то же, что и Dim;

**Public** — такая переменная будет доступна всем процедурам во всех модулях данного проекта, если вы объявили ее в области объявлений модуля. Если вы объявили ее внутри процедуры, она будет вести себя как Dim/Private;

**Static** — такие переменные можно использовать только внутри процедуры. Эти переменные видны только внутри процедуры, в которой они объявлены, зато они сохраняют свое значение между разными вызовами этой процедуры. *Обычно используются для накопления каких-либо значений.* Например:

```
Static nVar1 As Integer
nVar1 = nVar1 + 1
MsgBox nVar1
```

Объявление простых переменных имеет следующий синтаксис:

1.Dim <имя переменной1, имя переменной2,…> As <имя типа>

или

1.Dim <имя переменной1> As <имя типа1>, <имя переменной2> As <имя типа2>,<имя переменной3> As <имя типа3>, …

# Пример работы с переменными в VBA может выглядеть так:

```
Dim nMyAge As Integer
nMyAge = nMyAge + 10
MsgBox nMyAge
```

Правила выбора имен в VBA едины для многих элементов (переменные, константы, функции и процедуры и т.п.).

Имя:

- должно начинаться с буквы;
- не должно содержать пробелов и символов пунктуации (исключение — символ подчеркивания);
- максимальная длина — 255 символов;
- должно быть уникальным в текущей области видимости (подробнее — далее);
- зарезервированные слова (те, которые подсвечиваются другим цветом в окне редактора кода) использовать нельзя.

При создании программ VBA настоятельно рекомендуется определиться с правилами, по которым будут присваиваться имена объектам — соглашение об именовании.

Чаще всего используется так называемое венгерское соглашение:

- имя переменной должно начинаться с префикса, записанного строчными буквами. Префикс указывает, что именно будет храниться в этой переменной:
- str (или s) — String, символьное значение;
- fn (или f) — функция;
- c (или сделать все буквы заглавными) — константа;

- b — Boolean, логическое значение (true или false);
- d — дата;
- obj (или o) — ссылка на объект;
- n — числовое значение.

- имена функций, методов и каждое слово в составном слове должно начинаться с заглавной буквы:

MsgBox objMyDocument.Name

Sub CheckDateSub()

- Константы определяются заглавными буквами, между словами ставили подчеркивания:

COMPANY_NAME

**Константы** — еще один контейнер для хранения данных, но, в отличие от переменных, они не изменяются в ходе выполнения VBA-программы.

В VBA константы определяются при помощи ключевого слова Const:

Const COMP_NAME As String = “Microsoft”

При попытке в теле процедуры изменить значение константы будет выдано сообщение об ошибке.

Существует несколько типов констант.

***Именованные константы*** – константы, имеющие заданное имя; это имя имеет конкретное неизменяемое значение.

В отличие от переменной, необходимо всегда явно объявлять именованные константы, используя ключевое слово Const.

Следует помещать объявления констант на модульном уровне, чтобы у них была наибольшая область действия.

***Литеральные константы*** – это константы, записываемые непосредственно в код.

Правила написания литеральных констант **(String, Integer, Data, Boolean):**

- строковые константы должны быть заключены в двойные кавычки ("");
- пустая строковая константа (нулевая строка) обозначается двумя двойными кавычками, между которыми ничего нет ("");
- строковая константа должна вся находиться на одной и той же строке.

*Численные константы* могут содержать любой из численных типов VBA.

Правила написания численных констант:

- численные константы должны состоять только из числовых символов от 0 до 9;
- численная константа может начинаться со знака минус и содержать десятичную точку;
- можно использовать экспоненциальное представление для численных констант;
- никакие другие символы или знаки в численных константах не допускаются.

***Константы Date*** необходимо помещать между знаками фунта **(#).** Независимо от того, в каком из форматов записывается литеральная константа Date, VBA переформатирует эту константу для соответствия одному из двух следующих форматов – в зависимости от того, содержит ли константа Date информацию о времени:

#2/5/02 9:17:00 PM#

#2/5/02#

***Константы Boolean*** – существуют только две правильные константы типа Boolean: True и False.

Типизированные константы используются при явном задании типа константы. Объявление конкретного типа данных для константы может повысить точность вычислений. Для констант можно использовать типы данных Byte, Boolean, Integer, Long, Single, Double, Currency, Date, String.

Синтаксис:

1. Const имя_константы As type = value, name As type = value,…

где type – имя любого из типов данных VBA;

value – значение, присваиваемое константе.

**Пример:**

1. Const Pi As Double = 3.14

Объявление массива производится очень просто:

1.Dim MyArray (2) As Integer

Такой массив может хранить три целочисленных элемента. 2 — это верхняя граница массива (upper bound). Количество элементов, которое может хранить массив, — от 0 до верхней границы включительно.

Если вам хочется, чтобы нумерация элементов в массиве начиналась с 1, то в раздел объявлений модуля нужно внести команду

1. Option Base 1

Присвоить значение отдельному элементу массива (в нашем случае — первому) можно очень просто:

1.MyArray (0) = 100

А затем это значение можно будет извлечь:

1.MsgBox MyArray (0)

Массивы вполне могут быть многомерными:

1.Dim MyArray (4, 9)

В каждой строке многомерного массива удобно хранить данные, относящиеся к одному объекту (например имя сотрудника, уникальный номер, номер телефона). В VBScript в одном массиве может быть до 60 измерений.

Часто необходимы массивы динамические — те, размер которых можно изменять в ходе выполнения. Динамический массив объявляется следующим образом:

1.Dim MyArray () '- объявляем массив без верхней границы, эту строку можно ‘пропустить

2.ReDim MyArray (4) ' — изменяем размер массива

Команда ReDim не только изменяет размер массива, но и удаляет из него все старые значения. Чтобы старые значения сохранить, используется ключевое слово Preserve:

1.ReDim Preserve MyArray (7)

Однако если новый размер массива меньше, чем кол-во помещенных в него элементов, слово Preserve не поможет — часть данных все равно будет потеряна.

Массивы можно создавать и заполнять одновременно при помощи встроенной функции Array():

1. Dim MyArray
2. MyArray = Array(100, 200, 300, 400, 500)

Указывать размер массива необязательно — он будет автоматически настроен в соответствии с кол-вом передаваемых элементов.

Очистить массив можно командой Erase:

1.Erase MyArray

Массив фиксированной длины просто очищается, динамический массив рацианализируется — его придется инициализировать (определять размер) заново.

В динамических массивах часто не известно, сколько элементов в массиве. Для определения кол-ва элементов используется функция UBound() (если массив одномерный или вас интересует размер первого измерения, то измерение передавать не надо):

1. UBound (имяМассива [, измерение])

Как ни удивительно, но при программировании в VBA вам редко придется сталкиваться с массивами. Вместо них в объектных моделях приложений Office обычно используются коллекции.

**Оператор IF**

IF <условие> THEN
<действия при выполнении условия>
[ELSEIF <условие - n>
<действия при не выполнении условия и выполнении условия-n>]
[ELSE
<действия при не выполнении условия>]
END IF

## Оператор Select Case

Оператор Select Case идеально подходят для проверки одного и того же значения, которое нужно много раз сравнить с разными выражениями. Синтаксис

[Case <список значений 1>
<действия 1>]
[**Case** <список значений 2>
<действия 2>]
. . . . . .
[**Case** <список значений n>
<действия n>]
[**Case Else**
<Действия при иной ситуации>]
**End Select**

**Оператор IIF**

Оператор IIF (непосредственное если) представляется собой условную функцию, синтаксис которой:

IIF (условие, значение да, значение нет)

Например

A=IIF(B>0, 3, 0)

# Операторы ввода-вывода

Для организации ввода/вывода можно использовать диалоговые окна. Наиболее часто в программах VBA встречаются две разновидности диалоговых окон: окна сообщений и окна ввода.

Окно сообщения используется для предоставления информации пользователю. Можно задать вывод на экран окна сообщения, в котором пользователь должен щелкнуть на одной из кнопок, прежде чем продолжить работу.

**Синтаксис обращения к функции MsgBox:**

**MsgBox(prompt[, buttons] [, title] [, helpfile, context])**

Например: MsgBox(“Выполнено”)

Окно ввода предназначено для получения информации от пользователя. Окно ввода содержит поле ввода необходимой информации, сообщение, на которое пользователь должен отреагировать и кнопки **ОК** и **Отмена**.

Окно ввода создается и выводится на экран с помощью функции InputBox.

**Синтаксис вызова функции InputBox:**

**InputBox (prompt [, title] [, default] [, xpos] [, ypos] [, helpfile, context]).**

Например: A = InputBox(“Введите значение A=”)

# Циклы с параметром

Циклы используются в ситуациях, когда нужно выполнить какое-либо действие несколько раз.

Первая ситуация - когда известно, сколько раз нужно выполнить какое-либо действие, в этом случае используется конструкция For…Next:

For пЦикла = НачЗначение TO КонЗначение [Step Шаг]

[тело]

NEXT

Для безусловного выхода из конструкции For…Next используется команда Exit For.

## Цикл «Для каждого»

VBA предлагает специальную конструкцию "***For Each***" для организации циклов «для каждого», используемые для перебора элементов коллекций или массивов. Синтаксис конструкции таков:

For Each <Элемент> In <Группа>
[<Операторы>]
[Exit For]
[<Операторы>]
Next [<Элемент>]

<Группа> - имя объекта-коллекции или массива.

<Элемент> - объект (переменная), совпадающий по классу с элементами коллекции или имеющий тип Variant. Для массивов допустим только тип Variant.

## Цикл с предусловием

Критерием продолжения цикла является какое-то условие. Это условие помещается в начало (заголовок) цикла. Сначала проверяется условие, а потом уже выполняется тело цикла. Как правило такие циклы выполняются пока условие истинно. Как только условие становится ложным (False) выполнение цикла заканчивается. Тело данного цикла может ни разу не выполниться, если при первой проверке условия оно (условие) окажется ложным.

В VBA цикл с предусловием можно реализовать тремя способами при помощи следующих инструкций:

While <условие>
[выражения]
Wend

Do While <условие>
[выражения]
[Exit Do]
[выражения]
Loop

Do Until <условие>
[выражения]
[Exit Do]
[выражения]
Loop

## Цикл с постусловием

Оператор цикла с постусловием реализуется при помощи следующих двух инструкций

DO
[выражения]
[Exit Do]
[выражения]
Loop While <условие>

DO
[выражения]
[Exit Do]
[выражения]
Loop Until <условие>

Основное отличие цикла с постусловием от цикла с предусловием заключается в том, что сначала выполняется тело цикла, а уже затем проверяется условие, поэтому тело данного цикла хотя бы один раз выполнится обязательно.

Немедленный выход из цикла можно произвести по команде Exit Do.

В VBA предусмотрены следующие типы процедур:

Процедура типа Sub (подпрограмма) — универсальная процедура для выполнения каких-либо действий:

```
Sub Farewell()
MsgBox “Goodbye”
End Sub
```

Процедура типа Function (функция) — тоже набор команд, которые должны быть выполнены. Принципиальное отличие только одно: функция возвращает вызвавшей ее программе какое-то значение, которое там будет использовано.

Пример процедуры:

```
Function Tomorrow()
Tomorrow = DateAdd("d", 1, Date())
End Function
```

и пример ее вызова:

```
Private Sub Test1()
Dim dDate
dDate = Tomorrow
MsgBox dDate
End Sub
```

В тексте функции необходимо предусмотреть оператор, который присваивает ей какое-либо значение. В нашем случае это строка

Tomorrow = DateAdd(" d", 1, Date()).

## Области видимости процедур

По умолчанию все процедуры VBA (за исключением процедур обработки событий) определяются как открытые (Public). Это значит, что их можно вызвать из любой части программы — из того же модуля, из другого модуля, из другого проекта.

Public Sub Farewell()

или, поскольку процедура определяется как Public по умолчанию, то можно и так:

Sub Farewell()

Можно объявить процедуру локальной:

Private Sub Farewell()

В этом случае эту процедуру можно будет вызвать только из того же модуля, в котором она расположена. Такое решение иногда может предотвратить ошибки, связанные с вызовом процедур, не предназначенных для этого, из других модулей.

Можно ограничить область видимости открытых процедур (тех, которые у вас определены как Public) в каком-то модуле рамками одного проекта. Для этого достаточно в разделе объявлений этого модуля вписать строку

Option Private Module

Если при объявлении процедуры использовать ключевое слово Static, то все переменные в этой процедуре автоматически станут статическими и будут сохранять свои значения и после завершения работы процедуры. Пример:

```
Private Static Sub Farewell()
```

# Передача параметров

**Параметры** — значения, которые передаются от одной процедуры другой. В принципе, можно обойтись и без параметров, воспользовавшись только переменными уровня модуля, но при использовании параметров читаемость программы улучшается.

Чтобы процедура имела возможность принимать параметры, ее вначале нужно объявить с параметрами.

Например, вот пример простой функции, которая складывает два числа и выводит результат:

```
Function fSum (nItem1 As Integer, nItem2 As Integer)
fSum = nItem1 + nItem2
End Function
```

Вызов ее может выглядеть так:

MsgBox(fSum(3, 2))

В данном случае мы объявили оба параметра как обязательные, и поэтому попытка вызвать функцию без передачи ей какого-либо параметра (например, так: MsgBox (fSum(3))) приведет к ошибке "Argument not optional" — "Параметр не является необязательным".

Чтобы можно было пропускать какие-то параметры, эти параметры можно сделать необязательными. Для этой цели используется ключевое слово Optional:

Function fSum (nItem1 As Integer, Optional nItem2 As Integer)

В справке по встроенным функциям VBA необязательные параметры заключаются в квадратные скобки.

Для проверки того, был ли передан необязательный параметр, используется либо функция IsMissing (если для этого параметра был использован тип Variant), либо его значение сравнивается со значениями переменных по умолчанию (ноль для числовых данных, пустая строка для строковых и т.п.)

Вызов функции с передачей параметров может выглядеть так:

```
nResult = fSum (3, 2)
```

Однако здесь есть несколько моментов, которые необходимо рассмотреть.

В нашем примере мы передаем параметры по позиции, то есть значение 3 присваивается первому параметру (nItem1), а значение 2 — второму (nItem2). Однако параметры можно передавать и по имени:

nResult = fSum (nItem 1 := 3, nItem 2 := 2)

Конечно, вместо явной передачи значений (как у нас — 3 и 2) можно использовать переменные. Однако что произойдет с переменными после того, как они "побывают" в функции, если функция изменяет их значение? Останется ли это значение за пределами функции прежним или изменится?

Все зависит от того, как именно передаются параметры — **по ссылке** (по умолчанию, можно также использовать ключевое слово ByRef или **по значению** — нужно использовать ключевое слово ByVal).

Если параметры передаются по ссылке, то фактически в вызываемую процедуру передается ссылка на эту переменную в оперативной памяти. Если эту переменную в вызываемой процедуре изменить, то значение изменится и в вызывающей функции. Это — принятое в VBA поведение по умолчанию.

Если параметры передаются по значению, то фактически в оперативной памяти создается копия этой переменной и вызываемой процедуре передается эта копия. Конечно же, чтобы вы не сделали с этой копией, на исходную переменную это никак не повлияет и в вызывающей процедуре не отразится.

Продемонстрировать разницу можно на простом примере:

```
Private Sub TestProc ()
'Объявляем переменную nPar1 и присваиваем ей значение
Dim nPar1 As Integer
nPar1 = 5
'Передаем ее как параметр nItem1 функции fSum
MsgBox (fSum(nItem1:=nPar1, nItem2:=2))
'А теперь проверяем, что стало в нашей переменной nPar1, 'после того, как она побывала в функции fSum:
MsgBox nPar1
End Sub
Function fSum(nItem1 As Integer, nItem2 As Integer)
'Используем значение переменной
fSum = nItem 1 + nItem 2
'А затем ее меняем!
nItem 1 = 10
End Function
```

Проверьте, что будет, если поменять строку объявления функции

Function fSum(nItem1 As Integer, nItem2 As Integer)

на следующую строку :

Function fSum(byVal nItem1 As Integer, nItem2 As Integer)

Можно продемонстрировать компилятору VBA, что то, что возвращает функция, наш совершенно не интересует. Для этого достаточно не заключать ее параметры в круглые скобки. Например, в случае со встроенной функцией MsgBox это может выглядеть так:

```
MsgBox "Test"
```

а для нашей функции —

```
fSum 3, 2
```

Такой код будет работать совершенно нормально. Однако, если нам потребуется все-таки узнать, что возвращает MsgBox, то придется передаваемые ему параметры заключать в круглые скобки:

```
nTest = MsgBox("Test")
```

Для многих встроенных функций компилятор VBA в принципе не дает возможности игнорировать возвращаемое значение, заставляя помещать параметры в круглые скобки и принимать возвращаемое значение.