Лекция 1. Макрорекордер: начало автоматизации

**1.1. Макрорекордер — универсальный помощник**

Для автоматического создания *макросов* служит *макрорекордер*. Мы рассмотрим работу с ним на примере Microsoft Office *Word* 2007. Если отвлечься от *VBA* и говорить об обычной работе в *Word*, то окажется, что с помощью *макрорекордера* можно записывать действия, которые мы выполняем в программе. Например, записывается ввод и удаление текста, нажатие на кнопки вкладок, *форматирование* текста и так далее.

Однажды мне нужно было отформатировать довольно большой документ в кратчайшие сроки. А именно — выделить особыми стилями некоторые слова. Всего этих стилей было четыре, а это значит, что наиболее доступный метод офисной автоматизации — средство Формат по образцу — помогло бы не слишком сильно. Ручное форматирование (или создание новых стилей) так же казалось слишком медленным. Но на помощь пришли *макросы*, созданные с помощью макрорекордера. Назначив форматирование каждым из стилей собственному сочетанию клавиш, я значительно ускорил работу. А именно — правой рукой управлял мышью, перемещаясь по тексту и выделяя нужные слова, а левой нажимал необходимые сочетания клавиш.

*Алгоритм* работы с *макрорекордером* выглядит так:

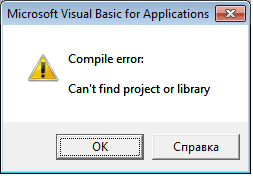
1. Включить запись *макроса*, настроив некоторые параметры;
2. Выполнить действия, которые вы хотели бы автоматизировать;
3. Остановить запись;
4. Использовать записанный макрос по необходимости.

*Макрорекордер*, помимо написания *макросов*, поможет вам лучше изучить возможности *VBA* и тонкости объектных моделей приложений Office (мы поговорим о них ниже). Например, вы не знаете точно, как именно реализовать то или иное действие программно, но знаете, как это делается вручную. Просто запишите *макрос* с нужными вам действиями, а потом откройте его в редакторе *VBA*. Вы сможете проанализировать полученный код, а возможно и включить его фрагменты в свое *приложение*.

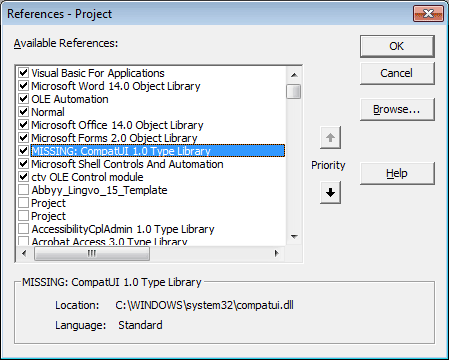
Лучший способ освоить *макрорекордер* — это попытаться самостоятельно записать *макрос*. Этим мы сейчас и займемся.

**1.2.Форматирование текста в MS Word**

**Примечание**. Если при попытке выполнить примеры этого курса вы, столкнетесь с ошибкой "Can't find project or library", перейдите в редактор Visual Basic, выполните команду **Tools > References**, в появившемся окне найдите те элементы списка, которые помечены как **MISSING**, снимите флаги напротив их имен, и если в списке имеются свежие версии отсутствующих библиотек, отметьте их, и нажмите **ОК**. После этого пример будет запускаться.



Can't find project or library



Tools-References

[01-01-Автоматическое форматирование.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/1/files/01_01.docm) — пример к п. 1.2.

Запишем в Microsoft *Word* *макрос*, который *по* нажатию комбинации клавиш форматирует выделенный текст определенным образом.

**1.2.1. Задание**

Создать макрос в Microsoft Word, который автоматически форматирует выделенный текст следующим образом:

* Шрифт: Times New Roman, 14-й, курсивный
* Цвет шрифта: красный

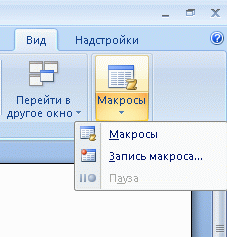
Назначить вызов *макроса* по нажатию комбинации клавиш **Alt + Ctrl + Shift + A** и по нажатию *настраиваемой кнопки*.

**1.2.2. Решение**

Выделим первый участок текста, который нужно отформатировать.

Прежде чем записывать *макрос*, важно очень точно спланировать свои действия. Если вы что-то сделаете не так во время записи, неправильные действия будут записаны в *макрос*. Например, выделять текст нужно до начала записи. Иначе в *макрос* попадет команда выделения текста и каждый раз после запуска программа будет выделять текст, а нам это не нужно.

Выберем вкладку ленты **Вид** и нажмем на ней направленную вниз треугольную стрелку под надписью **Макросы** ([рис. 1.1](https://www.intuit.ru/studies/courses/494/350/lecture/8354?page=1#image.1.1).).



**Рис. 1.1.**Начало записи макроса

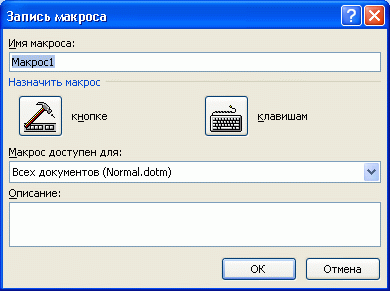
В открывшемся меню нам нужна команда **Запись макроса**.

Появится окно для настройки свойств *макроса* ([рис. 1.2](https://www.intuit.ru/studies/courses/494/350/lecture/8354?page=1#image.1.2).). Рассмотрим поля этого диалога.

**Имя макроса**: в это поле нужно ввести имя *макроса*. Имена *макросов* должны начинаться с буквы, не должны содержать пробелов. Желательно давать *макросам* какие-нибудь осмысленные имена. Например, Формат\_Times\_Красный.

**Назначить макрос кнопке**: диалоговое окно назначения *макроса* *настраиваемой кнопке*.

**Назначить макрос клавишам**: диалоговое окно назначения *макроса* сочетанию клавиш на клавиатуре.



**Рис. 1.2.**Окно настройки свойств макроса

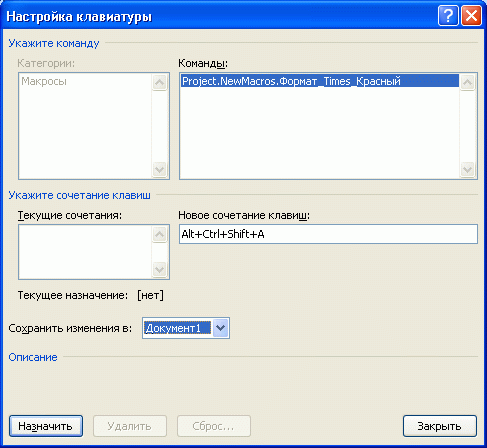
В поле **Макрос доступен для** надо выбрать место сохранения *макроса*, которое определяет его доступность для различных документов. По умолчанию здесь выбран параметр **Всех документов** ( Normal.dotm ).

Normal.dotm —это общий шаблон, доступный для всех документов Microsoft Word. Если макрос будет сохранен в Normal.dotm — вы сможете запустить его из любого Word-документа. Не следует сохранять все записываемые вами *макросы* в Normal.dotm. Делайте это лишь тогда, когда вы точно уверены в том, что макрос понадобится вам в различных документах. Мы выберем в этом поле значение **Документ 1 (документ)**. Именно так здесь называются еще не сохраненные документы. Выбрав этот пункт, мы сохраняем макрос в текущем документе, то есть сможем вызывать макрос лишь из этого документа.

В поле **Описание**содержится описание *макроса*. Вы вполне можете оставить его пустым.

Кнопка **OK** начинает запись *макроса*. Но до начала записи воспользуемся кнопкой **Назначить макрос клавишам**. Появится окно настройки комбинации клавиш для запуска *макроса* ([рис. 1.3](https://www.intuit.ru/studies/courses/494/350/lecture/8354?page=1#image.1.3).).

Чтобы назначить *макросу* сочетание клавиш, нужно установить курсор в поле **Новое сочетание клавиш** и нажать нужное сочетание на клавиатуре. В случае, если это сочетание не назначено ранее для быстрого вызова каких-либо команд, под полем **Текущие сочетания** вы увидите надпись **Текущее назначение: [нет]**. Если вы увидите здесь что-нибудь другое — лучше всего поискать свободное сочетание. Иначе вы можете столкнуться с неожиданным поведением знакомых вам горячих клавиш.

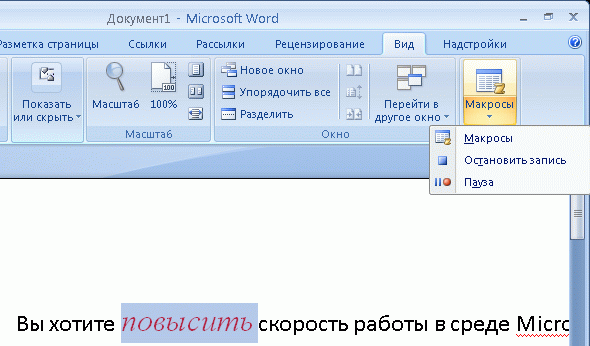


**Рис. 1.3.**Настройка сочетания клавиш для запуска макроса

В поле **Сохранить изменения в** выберем **Документ 1** — то есть наш документ, который мы в данный момент редактируем. По умолчанию здесь установлен уже знакомый вам Normal.dotm. Сохранять привязку сочетаний клавиш к *макросам* в Normal.dotm следует лишь в том случае, если вы, во-первых, на предыдущем шаге сохранили макрос в Normal.dotm, а во-вторых, хотите, чтобы данное сочетание работало во всех остальных документах MS Word, а не только в текущем документе.

Теперь нажмем на кнопку **Назначить**. Выбранное сочетание клавиш переместится в поле **Текущие сочетания**. Нажав кнопку **Закрыть**, мы завершаем настройку горячих клавиш для нашего будущего *макроса* и начинаем его запись.

В процессе записи *макроса* указатель мыши дополняется значком с изображением кассеты. Щелчок правой кнопкой не действует. Поэтому нам нужно отформатировать текст, пользуясь вкладками ленты Microsoft Word. Так и поступим. Настроим необходимые параметры форматирования, снова перейдем на вкладку **Вид** и нажмем там на кнопку **Остановить запись** ([рис. 1.4](https://www.intuit.ru/studies/courses/494/350/lecture/8354?page=1#image.1.4).)



**Рис. 1.4.**Остановка записи макроса

Обратите внимание на кнопку **Пауза**. С ее помощью можно приостановить запись *макроса*, выполнить какие-нибудь действия, которые не войдут в него, после чего возобновить запись.

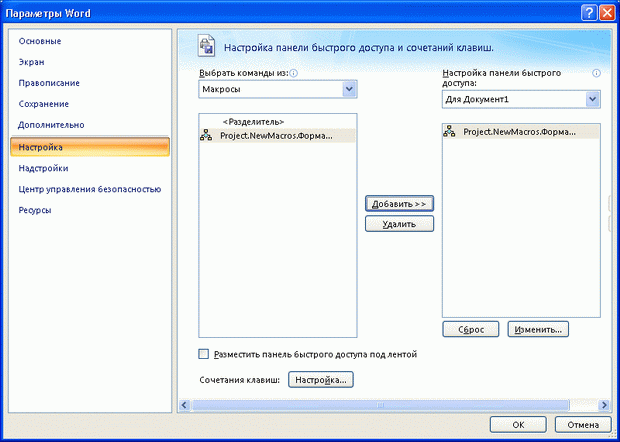
После записи *макроса* самое время его протестировать. Для этого выделим текст и нажмем **Ctrl + Alt + Shift + A**. Если все сделано правильно — текст будет отформатирован.

Но мы выполнили не все части задания. Как вы помните, нужно было настроить запуск *макроса* с помощью кнопки.

Для этого нажмите кнопку настройки панели быстрого доступа (она находится справа от панели) и в появившемся меню выберите пункт **Другие команды**. Появится окно для настройки команд Microsoft Word ([рис. 1.5](https://www.intuit.ru/studies/courses/494/350/lecture/8354?page=1#image.1.5).).

В списке **Выбрать команды из** выберите **Макросы**.

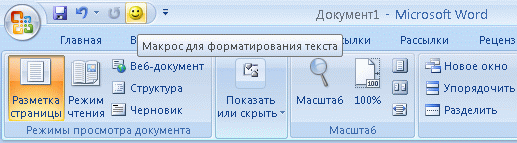
В поле, которое расположено ниже, появятся ссылки на доступные *макросы*. Выделите нужный и нажмите на кнопку **Добавить>>**. Ссылка на него появится в поле **Настройка панели быстрого доступа**. В списке настройки этого поля выберем документ, который мы в данный момент редактируем — тем самым мы внесем изменения лишь в панель быстрого доступа этого документа.

[](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/1/files/01_05.gif)

[увеличить изображение](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/1/files/01_05.gif)  
**Рис. 1.5.**Назначение макроса кнопке панели быстрого доступа

На этом можно завершить настройку. Однако, кнопка *макроса*, которая появится на панели быстрого доступа, будет выглядеть не слишком привлекательно. Поэтому выделим строку *макроса* в окне **Настройка быстрого доступа** и нажмем на кнопку **Изменить**. Появится окно настройки свойств кнопки *макроса*.

Здесь вам нужно выбрать понравившийся значок для кнопки *макроса* и ввести в поле **Отображаемое имя** текст, который будет появляться при наведении на кнопку вызова *макроса* указателя мыши. Вот, что получилось в итоге ([рис. 1.6](https://www.intuit.ru/studies/courses/494/350/lecture/8354?page=1#image.1.6).).



**Рис. 1.6.**Кнопка для быстрого вызова макроса на панели быстрого запуска

Сохраняя файл Microsoft Word, содержащий *макросы*, обязательно убедитесь в том, что вы сохраняете его в формате .docm (Документ Word с поддержкой *макросов* ). Дело в том, что используемый по умолчанию формат .docx (Документ Word) не поддерживает *макросы*. Если вы сохраните документ с *макросами* в таком формате, результаты работы будут утеряны.

Теперь вы научились созданию *макросов* с помощью *макрорекордера*. Уже сейчас вы можете автоматизировать некоторые часто выполняемые операции. Например, ввод с клавиатуры больших повторяющихся участков текста.

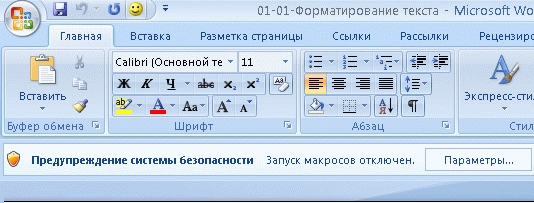
Чтобы использовать *макросы* — свои и тем более созданные другими — вам, возможно, понадобится изменить настройки безопасности Microsoft Office.

**1.3. Макросы и безопасность**

Microsoft Office снабжен собственной системой безопасности, основная задача которой — противодействие проникновению *макровирусов* — вредоносных программ на языке *VBA*.

*Макровирусы* — это основная проблема, которую породил *язык программирования*, встроенный в офисные программы. *Безопасность работы* с *макросами*сводится к очень простому правилу: не следует разрешать выполнение макросов, если вы не знаете точно, для чего нужны эти *макросы*.

Если вы открываете документ, который содержит подозрительный, с точки зрения системы, *макрос*, при настройках *безопасности**по* умолчанию этот *макрос* блокируется. В Microsoft *Word* выводится такое сообщение ([рис. 1.7](https://www.intuit.ru/studies/courses/494/350/lecture/8354?page=2#image.1.7).).

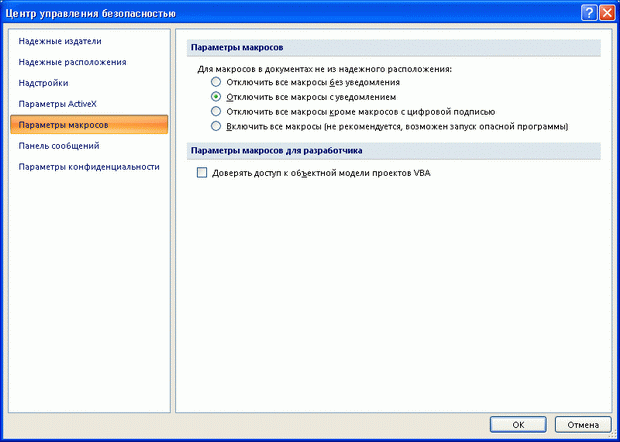


**Рис. 1.7.**Предупреждение системы безопасности

Если нажать на кнопку **Параметры**, которая находится в правой части панели сообщений, появится окно настройки свойств *макроса*. С помощью этого окна мы можем либо разрешить выполнение *макросов* в открытом документе, выбрав *пункт* **Включить это содержимое**, либо запретить, выбрав **Установить защиту от неизвестного содержимого**.

Для настройки системы безопасности в Microsoft *Word* нажмите на кнопку **Microsoft Office**, далее — на кнопку **Параметры Word**. В левой части появившегося окна выберите *пункт* **Центр управления безопасностью**. На странице центра управления безопасностью нажмите кнопку **Параметры центра управления безопасностью**. Появится окно настройки параметров *макросов* ([рис. 1.8](https://www.intuit.ru/studies/courses/494/350/lecture/8354?page=2#image.1.8).).

Практика показывает, что при интенсивном использовании собственных *макросов*, удобнее и безопаснее всего использовать *пункт*, установленный *по* умолчанию — **Отключить все макросы с уведомлением**. Даже если сообщения системы *безопасности*кажутся вам слишком назойливыми — не стоит поддаваться искушению и активировать *пункт* **Включить все макросы**. Ведь так вы ставите вашу систему под угрозу. Даже используя антивирусное *ПО*, нельзя забывать о потенциальной опасности *макросов*.

[](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/1/files/01_08.gif)

[увеличить изображение](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/1/files/01_08.gif)  
**Рис. 1.8.**Центр управления безопасностью: настройка безопасности макросов

Особый интерес представляет *пункт* **Отключить все макросы кроме макросов с цифровой подписью**. Активировав его, вы сможете спокойно запускать *макросы*, которые вы создали и подписали сами.

Подробности о цифровой подписи вы найдете во второй главе — там, где мы будем знакомиться с редактором *VBA*.

Теперь, когда мы обсудили *безопасность макросов*, давайте поговорим о *макросах* в Microsoft *Excel*.

**1.4. Копирование данных в MS Excel**

[01-02-Автоматическое копирование.xlsm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/1/files/01_02.xlsm) — пример к п. 1.4.

Процесс создания *макросов* в MS *Excel* очень похож на создание *макросов* в MS *Word*. Но есть здесь и свои тонкости. Мы создадим *макрос*, который *по* нажатию комбинации клавиш копирует значения выделенных пользователем ячеек в другие ячейки.

**1.4.1. Задание**

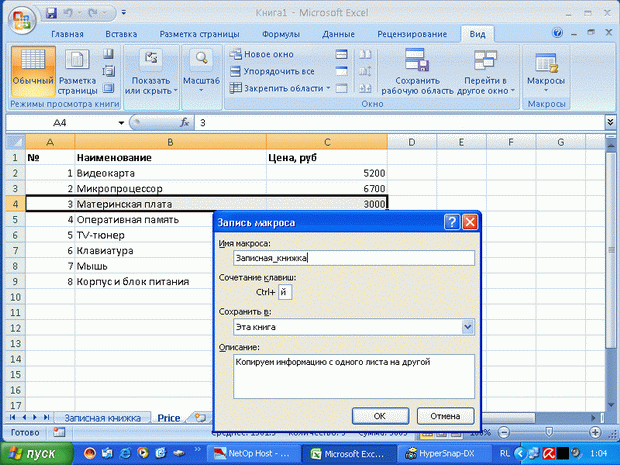
Создать макрос в Microsoft Excel, который предназначен для копирования выделенной строки из одного листа (пусть это будет некий прайс-лист) в текущую строку (начиная с выделенной ячейки) другого листа книги Excel. После работы *макроса* на экране должен быть отображен лист, с которого осуществлялось копирование.

Назначить вызов *макроса* по нажатию комбинации клавиш **Ctrl + Й**

Это задание можно рассматривать как создание автоматической "записной книжки" емкостью в одну запись.

**1.4.2. Решение**

Перед записью *макроса* выделим ячейки, из которых будем копировать данные. Далее перейдем на вкладку **Вид** и запустим с нее запись *макроса* ([рис. 1.9](https://www.intuit.ru/studies/courses/494/350/lecture/8354?page=2#image.1.9).).

[](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/1/files/01_09.gif)

[увеличить изображение](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/1/files/01_09.gif)  
**Рис. 1.9.**Начало записи макроса в MS Excel

Настроить быстрый вызов *макроса* можно в том же окне, где задается имя *макроса* и место сохранения. Имя Excel- *макросов* подчиняется тем же законам, что и в MS Word, а в качестве горячих клавиш предлагается использовать клавишу в комбинации с **Ctrl**.

После начала записи щелкнем правой кнопкой мыши по выделенной области и выберем в появившемся меню пункт **Копировать**. Перейдем на лист **Записная книжка** и вставим скопированное туда командой **Вставить** с вкладки **Главная**. Далее – вернемся на лист **Price** и остановим запись.

Сохраняя книгу Microsoft Excel, содержащую макросы, убедитесь в том, что сохраняете ее в формате "Книга Excel с поддержкой макросов", то есть итоговый файл имеет расширение .xlsm.

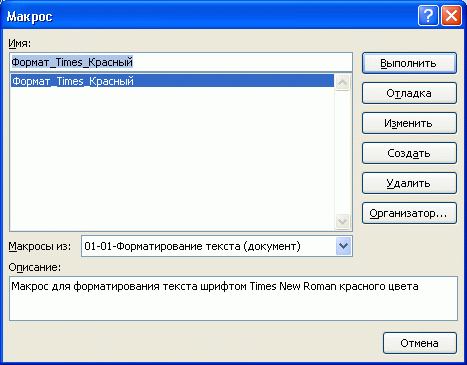
Если макрос MS Excel понадобится вам не только в одном документе — сохраните его в Личной книге макросов — эта опция доступна в окне записи макроса при выборе места сохранения.

Наш макрос работает. Но скажите, много ли от него пользы? "Записная книжка" емкостью в одну запись (ведь при копировании новых данных, старые попросту затираются) может найти практическое применение, но оно очень ограничено.

Для того, чтобы пользоваться возможностями приложений на полную мощность, не обойтись без полноценного программирования.

**1.5. Управление макросами**

Вкладка ленты **Вид** в Microsoft *Word* и Microsoft *Excel* содержит кнопку **Макросы**. Она расположена над раскрывающимся *меню*, которым мы пользовались до сих пор. Воспользуемся ею. Откройте один из документов MS *Word*, содержащий *макросы*, и нажмите на эту кнопку. Появится окно **Макрос** ([рис. 1.10](https://www.intuit.ru/studies/courses/494/350/lecture/8354?page=2#image.1.10).), которое предназначено для управления *макросами*.



**Рис. 1.10.**Окно управления макросами в MS Word

В данном случае в окне **Макрос** отображается *список* *макросов* из открытого документа. Чтобы посмотреть *макросы*, находящиеся в других местах (например, в Normal.dotm ), воспользуйтесь списком **Макросы из**.

Имя выделенного *макроса* (это тот самый *макрос*, который мы записали выше) отображается в строке **Имя**. Именно с этим *макросом* мы можем совершать различные действия в данном окне. В частности, здесь доступны следующие варианты:

**Выполнить** — выполняет *макрос*. Вместо того, чтобы пользоваться кнопкой быстрого вызова *макроса* или *настраиваемой комбинацией клавиш*, для выполнения *макроса* можно воспользоваться этой кнопкой. Это удобно для запуска *макросов*, которые используются очень редко.

**Изменить** — нажав на эту кнопку, вы откроете редактор *Visual* *Basic*, который позволяет изменять *макросы* и создавать собственные программы.

**Создать** — наберите в строке **Имя** желаемое имя нового *макроса* и нажмите на эту кнопку — система создаст новый пустой *макрос* и откроет его в редакторе.

**Удалить** — удаление *макроса*.

**1.6. Выводы**

Теперь вы можете создавать простые *макросы* и автоматизировать с их помощью некоторые *операции*. Но, думаю, вы уже чувствуете, что вам остался буквально один шаг от *макрорекордера* к самостоятельному созданию *макросов*. Предлагаю сделать этот шаг в следующей лекции.

Лекция 2. Начинаем программировать

**2.1. Редактор Visual Basic**

[02-01-Ввод текста.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/2/files/02_01.docm) - пример к п.2.1.

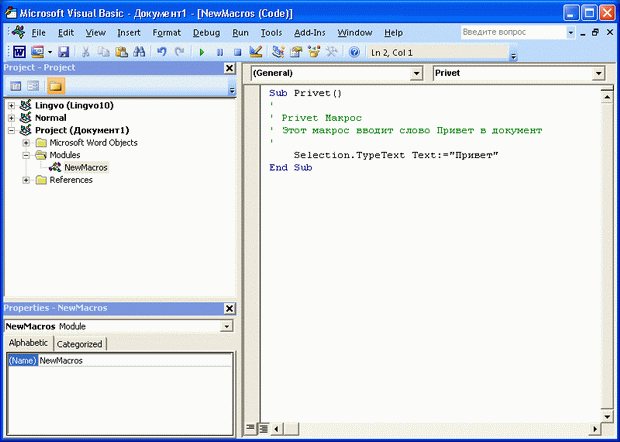
**2.1.1. Задание**

Записать в Microsoft Word макрос, который автоматически вводит в позицию курсора слово "Привет". Отредактировать макрос вручную таким образом, чтобы он выводил "До свидания".

**2.1.2. Решение**

Начнем запись макроса (Назовем его "Privet"), введем с клавиатуры слово "Привет", после чего остановим запись.

Откроем окно **Макросы**нажатием соответствующей кнопки на панели Вид, выберем наш макрос и нажмем на кнопку **Изменить**. Перед нами откроется окно редактора Visual Basic с открытым в нем макросом ([рис. 2.1](https://www.intuit.ru/studies/courses/494/350/lecture/8355?page=1#image.2.1).)

[](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/2/files/02_01.gif)

[увеличить изображение](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/2/files/02_01.gif)  
**Рис. 2.1.**Окно редактора Visual Basic

Окно редактора включает в себя несколько рабочих областей - окон, служащих для выполнения различных действий. На [рис. 2.1](https://www.intuit.ru/studies/courses/494/350/lecture/8355?page=1#image.2.1). в окне редактора присутствуют три рабочих области:

**Code (Код)** - это окно, в котором пишут тексты VBA-программ и редактируют макросы. Эта область расположена справа и занимает большую часть окна редактора.

**Project Explorer (Проводник Проекта)** - это окно обычно открыто в левой верхней части окна редактора. Оно отображает информацию о компонентах проекта, позволяет быстро перемещаться между компонентами.

**Properties (Свойства)** - отображает *свойства* выделенного объекта. Обычно располагается в левой нижней части рабочего окна.

Ниже перечислены другие рабочие области, которых мы коснемся в дальнейшем изложении.

**UserForm (Форма)** - служит для редактирования пользовательской формы в визуальном режиме.

**Toolbox (Панель элементов управления)** - содержит набор элементов управления (например - это кнопки, поля ввода и т.д.), которые можно добавлять на формы или в документы.

**Object Browser (Обозреватель объектов)** - служит для просмотра информации об объектах, доступных в данном приложении.

**Watch, Locals, Immediate** - окна, средства которых используются при отладке приложений.

Давайте изучим содержимое окна **Code**. Посмотрите на первую и последнюю строчку текста. Первая строчка выглядит так:

Sub Privet ()

Это - начало процедуры макроса. Процедура - это самостоятельный участок кода. Если хотите, можете считать процедуру отдельной программой. Думаю, вам знакомо название процедуры макроса - ведь мы сами дали ему это имя при записи.

Последняя строчка нашего макроса выглядит так:

End Sub

Получается, что все наши действия в ходе записи макроса описаны командами, находящимися между Sub и End Sub. А что же там находится?

Сразу после первой строки макроса можно увидеть несколько строчек, начинающихся со значка ' (апостроф). Он указывает на то, что текст данной строки - это комментарий. Комментарии в текстах VBA-программ автоматически выделяются зеленым цветом. В нашем случае комментарии содержат информацию об имени макроса и тот текст, который мы вводили в качестве комментария к макросу.

Дальше начинается самое интересное - код, который автоматизирует ввод текста. В нашем случае полезная часть макроса состоит всего из одной строчки. Вот она:

Selection.TypeText Text:="Привет"

Давайте попытаемся "перевести" текст этой программной строки на русский язык, учитывая, что Selection переводится как "выделение", Type - "печатать", Text - текст. Получается, что этой строкой мы даем компьютеру такую команду: "Напечатать в выделенной позиции текст "Привет"".

Как вы думаете, что нужно сделать с этой строчкой, чтобы программа вместо "Привет" вывела бы "До свидания"? Наверняка вы уже догадались, что для этого нужно лишь заменить текст, который расположен в кавычках.

В результате у нас получилась такая строка:

Selection.TypeText Text:="До свидания"

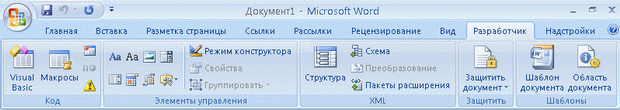
Многие начинающие программисты не придают значения точкам, запятым, скобкам и кавычкам, которые попадаются им в программе. Обычно это приводит к массе ошибок, которые вызывают у новичка недоумение. Поэтому будьте предельно внимательны, редактируя тексты программ.

Отредактируйте текст программы, закройте окно редактора (или, не закрывая его, перейдите в окно документа), и, установив курсор в желаемое место вставки текста, выполните макрос.

Примите мои поздравления. Первое знакомство с редактором Visual Basic и с языком VBA состоялось. Ваш первый вручную отредактированный макрос работает. Но сейчас у вас гораздо больше вопросов, чем ответов.

**2.2. Использование вкладки Разработчик**

В приложениях *Microsoft* Office предусмотрена специальная вкладка - **Разработчик**. Она служит для работы с *VBA*-программами, элементами управления, которые можно добавлять в документ и т.д. *По* умолчанию эта вкладка скрыта. Чтобы отобразить ее, например, в *Microsoft* *Word*, нажмите кнопку **Microsoft Office** (в левом верхнем углу окна программы), в появившемся окне нажмите на кнопку **Параметры Word**, после чего в окне **Параметры Word** установите галочку в *поле* **Показывать вкладку Разработчик на ленте**. Вот как выглядит эта вкладка ([рис. 2.2](https://www.intuit.ru/studies/courses/494/350/lecture/8355?page=1#image.2.2).).

[](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/2/files/02_02.gif)

[увеличить изображение](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/2/files/02_02.gif)  
**Рис. 2.2.**Вкладка Разработчик на ленте Microsoft Word

Обратите внимание на кнопку **Visual Basic**- она расположена в области **Код**. Нажав на эту кнопку, вы запустите редактор *VBA*. В области **Элементы управления** расположены миниатюры элементов управления, которые можно располагать в документах.

*Если вы включите отображение вкладки Разработчик - в строке состояния редактора появится кнопка, с помощью которой можно быстро начать или остановить запись макроса*.

В дальнейшем мы постоянно будем пользоваться элементом управления **Кнопка** для демонстрации различных возможностей *VBA*.

**2.3. Объектно-ориентированное программирование**

В последние годы огромную популярность приобрело так называемое *объектно-ориентированное программирование* ( *ООП* ). Для того, чтобы эффективно работать в *VBA*, нужно понимать основные положения *ООП*. В частности, вам нужно разобраться, что такое *классы*, *объекты*, что такое *методы*, *свойства* и *события* *объектов*.

***Класс - это шаблон для создания объектов***. Например, пусть у нас есть *класс*"автомобиль". Он создан *по* общему чертежу, *по* единой проектной документации. Но каждый отдельный *объект* - конкретный экземпляр автомобиля -индивидуален. Например, они различаются цветом, особенностями отделки салона и так далее. Точно также происходит и при работе с программными *классами* и *объектами*. Один *класс* может породить множество различных *объектов*.

На первых этапах VBA-программирования вам не придется самостоятельно создавать *классы*, но вы постоянно будете обращаться к *объектам* из своих процедур. Однако знайте, что VBA в MS Office 2007 поддерживает написание полностью объектно-ориентированных приложений *классов* и инструментария работы с ними.

*Объект* - это нечто целостное, неделимое, содержащее в себе данные и код. В *объекте* может храниться *информация* и программные инструкции для обработки этой информации.

У *объектов* есть *свойства* и *методы*. Объекты могут откликаться на *события*. Чтобы объяснить эти важнейшие понятия, разовьем уже использованный пример.

Напомним: наш *объект* - это автомобиль. Какие *свойства* есть у автомобиля? Цвет, тип двигателя, скорость движения и так далее. А какие *методы*? Что наш автомобиль умеет делать? Например, передвигаться, ехать. Получается, что у автомобиля есть как *минимум* два *метода*: "начать движение" и "остановиться". Точно так же, *свойства* и *методы* есть и у *объектов*, которыми мы будем пользоваться в *VBA*. А как насчет *событий*? Например, *событием* *по* отношению к автомобилю может быть нажатие педали газа, которое приводит к увеличению скорости.

Детали работы *объекта* скрыты от пользователя. Обычно пользователю доступны лишь те *свойства* и *методы*, которые специально для этого предназначены. *Пользователь* не может повлиять на внутренние *механизмы* *объекта*, работая лишь со специально выделенными для взаимодействия с внешним миром *методами* и *свойствами*.

Обращение к *методам* *объектов* в *VBA* ведется таким образом:

Объект.Метод

Возможны и другие варианты использования *методов*. Например, если в *метод* нужно передать какой-то *параметр*, его вызов будет выглядеть так:

Объект.Метод (Параметры)

Обратите внимание на то, что *объект* и *метод* отделены точкой. Если вы вводите в редакторе *VBA* имя *объекта* и ставите после него точку - редактор подскажет вам, какие *методы* и *свойства* есть у этого *объекта*.

С помощью *методов* *объекта* мы даем *объекту* команды. Например, такая *команда*: Application.Quit, приводит к закрытию приложения. Очевидно, что Application - это *объект* "*Приложение*", а Quit - это *метод* "Выйти".

Со *свойствами* обычно работают так:

Объект.Свойство=Значение

Такой командой мы присваиваем некое *значение* *свойству* *объекта*.

А так: Переменная=Объект.Свойство можно присвоить некой *переменной* *свойство* *объекта*. Подробнее с *переменными* мы познакомимся ниже. Пока можете считать *переменные* специальными "ящиками" в памяти компьютера, в которых можно хранить различные данные.

Свойства - это, как мы уже говорили, признаки *объектов*. Например, такая *команда*, txt\_User\_Name.Text="Александр", выведет в текстовом *поле* ( *объект* txt\_User\_Name ) *слово* "Александр". Очевидно, что эта *команда* модифицирует *свойство* *объекта* Text, которое отвечает за выводимый на *объекте* текст.

Если вам пока непонятно, что это за текстовые поля и их *свойства* - просто читайте дальше. В следующей главе мы подробно все это обсудим - и вы сможете соединить теорию, которую вы уже знаете, с практикой.

*Событие* - это действие, которое *объект* распознает и на которое реагирует. Например, *событием* может быть щелчок мышью *по* кнопке, открытие нового документа и т.д.

Объекты могут объединяться в *коллекции объектов*. Фактически, *коллекции* - это *объекты*, которые содержат в себе другие *объекты*. Обычно в *коллекцию* входят *объекты*, имеющие один и тот же тип - то есть - одинаковые *объекты*.

В следующей главе мы вплотную займемся работой с *объектами*, а пока обсудим очень важную тему - объектные модели приложений Office.

**2.4. Введение в объектные модели приложений Office**

*Объекты* приложений Office объединены в иерархическую структуру, которую называют объектной моделью. У каждого приложения своя *объектная модель*, то есть свой набор *объектов*. Например, в *Microsoft* *Word* на верхнем уровне иерархии стоит библиотека *классов* Word. Она включает в себя множество *классов*, например - Application, который символизирует *приложение* *Microsoft* *Word*. Application, в свою *очередь*, включает в себя другие *объекты*, *методы*, *свойства*. Например, такая *команда* приводит к закрытию *Microsoft* *Word*:

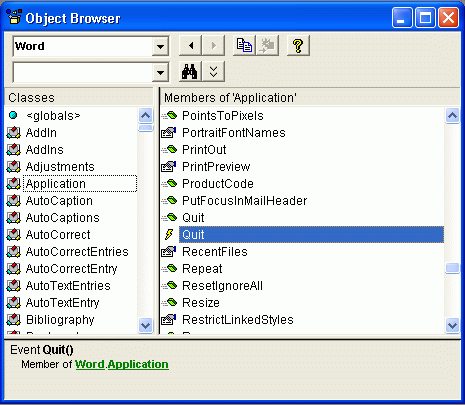
Word.Application.Quit

Обычно при работе в MS *Word* опускают вызов *объекта* Word, то есть вышеприведенную команду вполне может заменить такая:

Application.Quit

Вызов *объекта* Application тоже можно опускать, но делать это допустимо не всегда. Например, команду Application.Quit сократить уже не получится

Как мы уже говорили, в редакторе *VBA* есть средство для просмотра информации об *объектах* - **Object Browser**. Для его вызова служит одноименная кнопка на панели инструментов, или клавиша **F2** ([рис. 2.3](https://www.intuit.ru/studies/courses/494/350/lecture/8355?page=2#image.2.3).).



**Рис. 2.3.**Окно Object Browser

В левой части окна (она называется **Classes** ) находится *список* *классов* и перечислений, входящих в библиотеку *Word* (или в ту библиотеку, которая выбрана в *поле* **Object Browser** ), в окне **Members of** можно просмотреть *методы*, *свойства* и *события* выбранного в предыдущем окне *класса*.

Значок с изображением руки в этом окне означает *свойство*, значок молнии - *событие*, значок с изображением маленького предмета с черточками - *метод*.

Здесь нет информации о том, что делает тот или иной *метод*, что означает то или иное *свойство* и т.д. Как же узнать подробности? Для начала - задуматься над именами *классов*, *объектов*, свойств, *методов*, *событий*. Правда, для их понимания нужно знать значения английских слов, которые использованы для их именования.

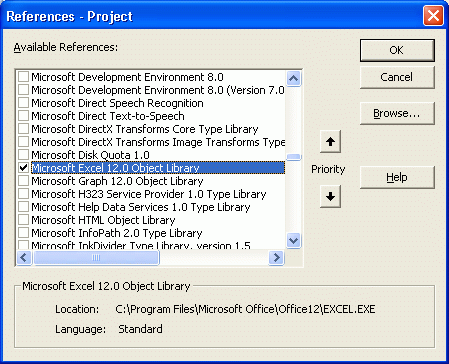
Далее, можно поискать информацию *по* интересующим вас *объектам* в справочной системе *VBA*, а так же - в базах данных *Microsoft* ([www.microsoft.com](http://www.microsoft.com/)).

Если вы не знаете, какой именно *объект* нужен для выполнения того или иного действия, но знаете как выполнить это действие с помощью визуальных средств приложения (например - с помощью кнопок на вкладках), вы можете исследовать объектные модели с помощью макрорекордера.

Объектные модели приложений огромны. С одной стороны, для эффективного использования всех *объектов*, нужно хорошо в них разбираться, с другой же - запомнить все это многообразие очень сложно. К счастью, для повседневной работы вам достаточно знать лишь основные *классы* и *объекты* объектных моделей, а если вам понадобится что-нибудь особенное - всегда можно воспользоваться справкой или использовать *макрорекордер*.

Помимо обычных объектных моделей приложений Office, вы можете использовать в своих программах другие объектные модели. Например, программируя для *Word*, можете воспользоваться объектной моделью *Excel* а так же - любыми другими моделями, установленными в системе. Для этого нужно подключить нужные модели из редактора *VBA* с помощью команды *меню* **Tools -> References**(Инструменты o Ссылки) ([рис. 2.4](https://www.intuit.ru/studies/courses/494/350/lecture/8355?page=2#image.2.4).).

В данном случае мы подключаем библиотеку *Microsoft* *Excel* (она ценна встроенными функциями) к *Microsoft* *Word*, то есть сможем пользоваться некоторыми функциями *Excel* в *Word*. Подключенные модели можно просматривать в окне *Project* Explorer.



**Рис. 2.4.**Подключение библиотеки объектов Microsoft Excel

Подключив объектную модель *Microsoft* *Excel* в *Microsoft* *Word*, вы сможете программно работать с документами MS *Excel* из программ, написанных в MS *Word* (и наоборот). Если вы собираетесь работать с базами данных - вам понадобится особая *объектная модель* *ADO* и т.д.

**2.5. Старые и новые версии Office: совместная работа**

Сегодня нередко встречается ситуация, когда пользователям приходится одновременно работать со старыми версиями MS Office (например, Office XP, 2003) и новыми (Office 2007). Как вы знаете, эти версии имеют различные форматы файлов. Например, в предыдущих версиях MS *Word* используется универсальный формат .doc для хранения файлов документов, .dot - для шаблонов, а в новой - .docx и .docm для документов (второй формат поддерживает макросы, а первый - нет), и . dotm - для шаблонов.

Новый Office нормально работает со старыми версиями файлов, в том числе и с файлами, содержащими макросы. Однако, если вы хотите, чтобы, скажем, ваши \*.doc - файлы, содержащие макросы, гарантированно работали в Office 2007 - обязательно тестируйте их в среде Office 2007.

Старые версии в исходном виде не умеют работать с новыми файлами. Чтобы они смогли читать новые файлы, нужно установить специальный конвертер, который можно скачать с сайта *Microsoft*, однако это не гарантирует полную совместимость программного кода. Так, объектные модели приложений Office 2007 дополнены новыми *объектами* (например, OMath для работы с формулами в MS *Word*).

Если вам приходится работать в среде, которая предусматривает параллельное использование двух разных версий Office, лучше всего создавать и отлаживать программы именно в той версии, в которой вы планируете их использовать.

Большая часть материала данной книги справедлива для всех современных версий Office, однако, как уже было сказано, объектные модели приложений различаются, поэтому, используя тот или иной *объект* в устаревших версиях - обязательно сверяйтесь со справочной системой или исследуйте *операции* с помощью макрорекордера.

**2.6. Выводы**

В этой лекции вы узнали самое главное о программировании на *VBA*. Теперь вы знаете, как найти редактор *Visual* *Basic* и где хранится код макроса. Теперь вам нужно научиться пользоваться редактором, ориентироваться в его инструментах. Всему этому посвящена следующая лекция.

Лекция 3. VBA-редактор

**3.1. Особенности главного меню VBA-редактора**

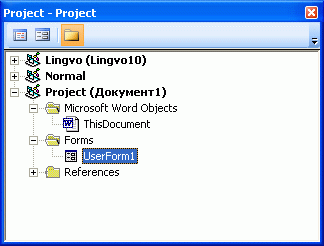
Окно редактора *Visual Basic* выглядит одинаково во всех приложениях Microsoft Office. Давайте рассмотрим главное *меню* этого окна, описав назначение наиболее часто используемых пунктов.

* **File** (Файл) — служит для работы с файлами.
* **File • Save**— сохраняет файл.
* **File • Import File** — позволяет импортировать внешний файл в редактор. Например, таким образом можно добавить в свой проект модуль (то есть — программный код) или форму.
* **File • Export File** — экспортирует данные из редактора во внешний файл. Например, этой командой можно сохранить редактируемую форму и передать ее другому разработчику.
* **File • Close and Return To Microsoft Word** — закрывает *VBA*-редактор и возвращается в Microsoft Word (аналогичная команда есть и для *MS Excel*). Вы можете просто переключаться между редактором и основным приложением в Панели задач Windows, не закрывая редактор.
* **Edit** (Правка) — содержит команды для правки. Помимо стандартных команд отмены и возврата последнего действия ( **Undo, Redo** ), вырезания, копирования, вставки ( **Cut, Copy, Insert** ), поиска ( **Find** ), это меню содержит несколько особенных команд. В частности, это **List Properties/Methods** (Список Свойств/Методов) и другие.
* **View**(Вид) — cодержит команды для отображения различных окон редактора. Названия команд соответствуют названиям окон.
* **Insert** (Вставка) — служит для вставки в проект форм ( **Insert • User Form** ), модулей ( **Insert • Module** ), процедур ( **Insert • Procedure** ), файлов ( **Insert • File** ) и модулей класса ( **Insert • Class Module** ). Чаще всего вам придется вставлять в проект формы. Эти команды продублированы на панели инструментов редактора.
* **Format** (Форматирование) — служит для управления расположением элементов управления на формах.
* **Debug** (Отладка) — содержит команды, позволяющие отлаживать программы.
* **Run** (Запуск) — содержит команды для управления выполнением программ. В частности, команда **Run • Sub/User Form** (Запуск • Процедура/Форма) запускает на выполнение активную процедуру или форму (рядом с этой командой стоит характерный зеленый треугольник). Команда **Run • Break** (Запуск • Приостановить) — приостанавливает выполнение программы, команда **Run • Reset**(Запуск • Перезапуск) – останавливает выполнение программы. Эти команды продублированы на панели инструментов редактора в виде кнопок с соответствующими пиктограммами.
* **Tools** (Инструменты) — содержит средства для настройки свойств редактора, подключения дополнительных библиотек объектов.
* **Add-Ins** (Дополнения) — позволяет управлять дополнениями. По умолчанию это меню содержит лишь одну команду, запускающую менеджер дополнений.
* **Window** (Окно) — стандартные команды для работы с окнами.
* **Help** (Помощь) — помощь по *VBA*.

В дальшейнем мы подробно рассмотрим многие из команд главного *меню*. А сейчас давайте поговорим об окне **Project** — вам постоянно придется пользоваться им.

**3.2. Подробности об окне Project**

Окно *Project* служит для навигации *по* проекту ([рис. 3.1](https://www.intuit.ru/studies/courses/494/350/lecture/8357?page=1#image.3.1).).



**Рис. 3.1.**Окно Project

Каждый открытый документ представляет собой отдельный проект. На [рис. 3.1](https://www.intuit.ru/studies/courses/494/350/lecture/8357?page=1#image.3.1). вы можете видеть окно **Project** для только что созданного документа MS *Word*. В проект добавлена форма.

Проект имеет иерархическую структуру. Так, в разделе **Microsoft Word Objects** (Объекты Microsoft *Word*) мы можем видеть *объект* **ThisDocument** (Этот документ), то есть документ, который мы создали. В разделе **Forms** (Формы) мы можем видеть *объект* **UserForm1** — только что вставленную в проект форму.

Чтобы отобразить *объект*, достаточно сделать *по* нему *двойной щелчок* или нажать на кнопку **View Object** (вторая слева в верхней части окна **Project** ). Чтобы просмотреть код объекта надо нажать на кнопку **View Code** (крайняя левая кнопка).

Щелкнув правой кнопкой мыши *по* названию проекта, можно увидеть его контекстное *меню*. Это *меню* содержит наиболее часто используемые команды для работы с проектами. В частности, особый интерес представляет *команда* **Project Properties** (Свойства проекта).

В окне **Project Properties** обратите внимание на вкладку **Protection** (Защита). Она позволяет защитить проект от просмотра и закрыть его паролем.

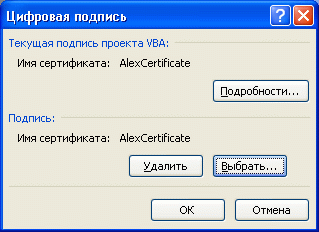
Для защиты проекта от просмотра установите галочку в *поле* **Lock project for viewing**(Закрыть проект от просмотра), для закрытия проекта паролем, введите в поля **Password**(*Пароль*) и **Confirm** **password** (Подтвердить *пароль*) *пароль*.

Существует специальное ПО, которое позволяет восстанавливать забытые пароли к *VBA*-проектам. Как вы понимаете, оно может помочь получить доступ в проект не только законным владельцам, но и злоумышленникам. Поэтому если вы, например, передаете кому-то ценный *VBA*-проект по слабо защищенным каналам связи — лучше зашифруйте весь файл с помощью какой-нибудь программы шифрования.

Еще одна полезная возможность, связанная с управлением проектами – подписывание проектов.

Прежде чем подписать проект, нужно создать *цифровой сертификат* для проектов *VBA*. Чтобы создать его, пройдите *по* пути **Пуск • Все программы • Microsoft Office • Microsoft Office Tools • Digital Certificate for VBA Projects**. Введите в появившемся окне имя для сертификата.

Чтобы теперь подписать проект, выделите его в окне *Project* и воспользуйтесь командой **Tools • Digital Signature**. В появившемся окне нажмите на кнопку **Выбрать** — появится окно для выбора сертификата. Выберите из этого окна интересующий вас сертификат. В нашем случае здесь присутствует лишь один, заранее созданный, сертификат. Вот как выглядит окно цифровой подписи после завершения выбора ([рис. 3.2](https://www.intuit.ru/studies/courses/494/350/lecture/8357?page=1#image.3.2).)



**Рис. 3.2.**Цифровая подпись проектов

Учтите, что автоматически создаваемый сертификат считается надежным лишь на том компьютере, в *хранилище сертификатов* которого он находится. Если вы подпишите свои проекты, вы сможете один раз разрешить их использование, выбрав соответствующий *пункт* в окне настройки *безопасности макросов*.

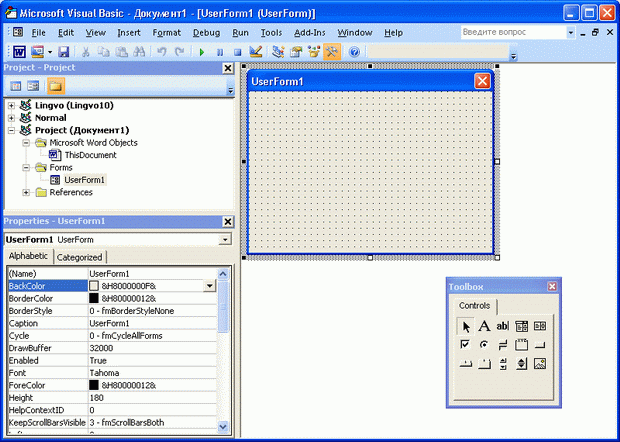
Это позволит, с одной стороны, не отвлекаться на постоянные запросы системы безопасности, а с другой — не понижать *безопасность системы*, разрешая выполнение всех макросов.

Теперь займемся работой с формами и элементами управления

**3.3. Формы и элементы управления**

[03-01-Формы и элементы управления.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/3/files/03_01.docm) — пример п. 3.3. — 3.6

Создайте новый документ Microsoft *Word* и откройте редактор *VBA*. Вставьте в проект новую форму командой **Insert • User Form** (Вставка • Форма). Вот что должно получиться ([рис. 3.3](https://www.intuit.ru/studies/courses/494/350/lecture/8357?page=1#image.3.3).).

[](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/3/files/03_03.gif)

[увеличить изображение](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/3/files/03_03.gif)  
**Рис. 3.3.**Новая форма в окне редактора VBA

Как видите, форма — это пустая заготовка будущего диалогового окна вашего приложения.

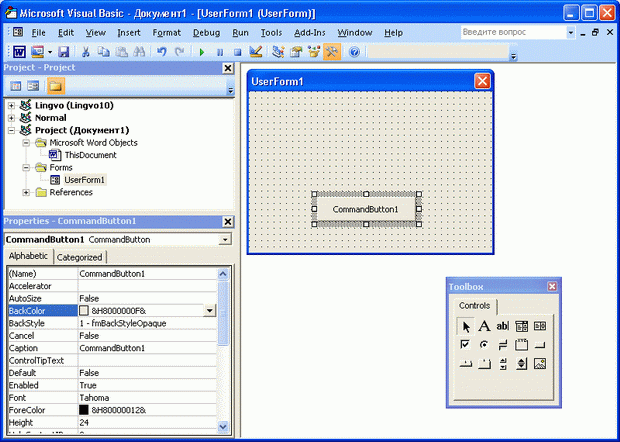
*По* краям новой формы вы можете видеть маркеры выделения. С их помощью можно настраивать ее размеры.

Обратите внимание на окно **Properties** (Свойства). Как вы уже знаете, оно содержит свойства объектов. Для выбора объекта, свойства которого будут отображены в этом окне, нужно либо выделить *объект*, либо указать его в списке, который находится в верхней части окна **Properties**.

Окно **Toolbox** (*Элементы управления*) содержит *элементы управления*, которые можно добавлять на форму. *Элементы управления* — это все то, что служит для управления работой программ. Например: кнопки, поля ввода, полосы прокрутки и так далее.

Для того, чтобы добавить элемент управления на форму, нажмите в окне **Toolbox** кнопку с изображением этого элемента и выделите мышью ту область на форме, которую он должен занять

Нажмем на кнопку **Toolbox'а** с изображением кнопки и добавим новую кнопку на форму. Вот, что у нас получилось ([рис. 3.4](https://www.intuit.ru/studies/courses/494/350/lecture/8357?page=1#image.3.4).).

[](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/3/files/03_04.gif)

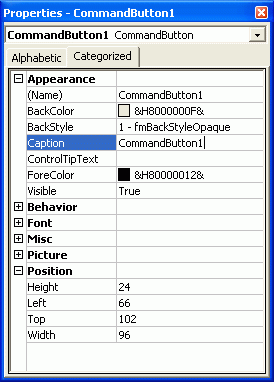
[увеличить изображение](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/3/files/03_04.gif)  
**Рис. 3.4.**Кнопка, добавленная на форму

Вокруг кнопки вы можете видеть маркеры выделения, которые служат для изменения ее размера. "Схватив" элемент управления мышью, вы можете перемещать его *по* форме.

**3.4. Окно Properties — управление свойствами**

Выделите кнопку и посмотрите на окно **Properties**. Теперь в нем отображаются ее свойства.

Свойства элементов управления могут быть отображены в двух разных представлениях. Первое – это алфавитный *список* (вкладка **Alphabetic** ) второе – *отображение* *по* категориям ( **Categorized** ). Начинающим обычно удобнее пользоваться видом свойств объекта *по* категориям ([рис. 3.5](https://www.intuit.ru/studies/courses/494/350/lecture/8357?page=2#image.3.5).). *Объединение* сходных свойств в группы позволяет легче разбираться с их предназначением.



**Рис. 3.5.**Вид свойств кнопки по категориям

На вкладке **Categorized** свойства объектов объединены в группы. Для удобства группы можно сворачивать и разворачивать. Давайте рассмотрим особенности различных групп, отображаемых в этом окне.

* **Appearance** (Внешний вид) — отвечает за *отображение объекта*, за надписи на нем, за его имя.
* **Behavior** (Поведение) — отвечает за поведение объекта. Например, за отображение многострочного текста.
* **Font** (Шрифт) — содержит свойство, отвечающее за шрифт, которым сделаны надписи на объекте.
* **Misc** (Разное) — различные настройки. Например — настройки указателя мыши, когда он будет находиться над объектом.
* **Picture** (Изображение) — информация об изображении, которое может отображаться на объекте.
* **Position** (Расположение) — определяет размер и положение объекта.

У форм есть *группа* свойств **Scrolling** (Скроллинг) — свойства этой группы управляют прокруткой формы.

Давайте рассмотрим свойства, общие для практически всех объектов — форм, различных элементов управления.

**3.5. Важнейшие свойства элементов управления**

**Name**(Имя) — очень важное свойство. *По* имени к объекту можно обратиться из программного кода.

Желательно давать объектам осмысленные имена — так гораздо легче пользоваться ими при разработке программ. Следите за тем, чтобы имена ваших объектов состояли лишь из цифр, латинских букв, символов подчеркивания. Если вы дадите объекту имя, например, содержащее знак "/", редактор не позволит использовать такое имя. При именовании объектов можно пользоваться и русскими буквами, но лучше этого не делать.

Переименуем форму в frm\_Main, а кнопку – в cmd\_First. Один из стилей именования объектов (так называемая венгерская *нотация*) заключается в использовании префикса, указывающего на *тип объекта*.

В [табл. 3.1](https://www.intuit.ru/studies/courses/494/350/lecture/8357?page=2#table.3.1). вы можете найти префиксы, которые можно использовать при именовании различных элементов управления.

|  |  |
| --- | --- |
| Таблица 3.1. Префиксы имен элементов управления | |
| **Элемент управления** | **Префикс** |
| UserForm (Форма) | frm |
| CommandButton (Кнопка) | cmd |
| TextBox (Текстовое поле) | txt |
| Label (Надпись) | *lbl* |
| ListBox (Список) | lst |
| ComboBox (Поле со списком) | cbo |
| ScrollBar (Полоса прокрутки) | *scr* |
| SpinButton (Счетчик) | spn |
| OptionButton (Радиокнопка) | *opt* |
| CheckBox (Флажок) | chk |
| ToggleButton (Выключатель) | tgl |
| Frame (Рамка) | fra |
| Image (Картинка) | img |
| MultiPage (Набор страниц) | *mlt* |
| TabStrip (Набор вкладок) | tab |

Вы можете использовать свой собственный стиль именования, главное — это чтобы имена были понятными и сразу указывали на назначение объекта.

Caption (Надпись) — определяет надпись, которая видна на элементе управления. Изменим свойство Caption для объекта frm\_Main на "Моя первая *программа*", а для объекта cmd\_First — "Нажми меня!".

Width (Ширина), Height (*Высота*) — определяют ширину и высоту объекта, которые измеряются в пикселях.

Font (*Шрифт*) — при попытке редактирования этого свойства открывается окно настройки параметров шрифта, в котором можно выбрать гарнитуру, размер, начертание шрифта.

ForeColor (Цвет шрифта) — задает цвет шрифта элемента управления.

BackColor (*Цвет фона*) — позволяет задать *цвет фона* элемента управления.

У кнопок и других элементов управления есть специфические свойства. В их число входят:

Top (*Расстояние* от верха формы), Left (*Расстояние* от левого края формы) — задают положение элемента на форме.

Visible (Видимость) — может принимать значения True (*Истина*) — то есть *объект* будет видимым, и False (*Ложь*) — *объект* будет невидимым.

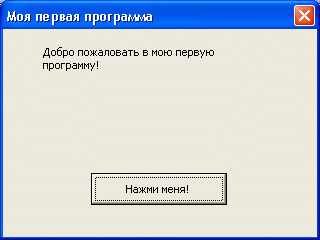
Enable (Включено) — если это свойство установлено в True (Истина) — объектом можно пользоваться. Если в False (Ложь) — *объект* будет неактивен. Например, кнопку "Далее" можно сделать неактивной до тех пор, пока *пользователь* не заполнит все обязательные поля ввода данных.

Некоторых других свойств объектов мы коснемся в дальнейшем.

Продолжим знакомство с элементами управления. Сейчас нас интересует элемент управления Label (*Метка*). Метки используются для размещения текстовой информации на формах. Добавьте в вашу форму новую метку (если сомневаетесь в названии элемента управления — наведите на него *указатель* мыши в панели инструментов — через некоторое время появится подсказка), дайте ей имя lbl\_First и разместите на ней надпись "Добро пожаловать в мою первую программу!". Чтобы разместить такую надпись на метке, можно модифицировать ее свойство Caption. Но можно поступить *по*-другому — выделить элемент управления и щелкнуть в том его месте, где расположен текст и отредактировать его прямо на элементе управления.

Мы еще не написали ни единой строчки кода, но уже можем запустить нашу первую программу. Щелкните *по* созданной форме и нажмите зеленую кнопку на панели инструментов редактора *VBA* (или выберите команду *меню* **Run • Sub/User Form**, можете вместо всего этого просто нажать клавишу **F5** ).

В результате вы увидите только что созданную форму во всей красе ([рис. 3.6](https://www.intuit.ru/studies/courses/494/350/lecture/8357?page=2#image.3.6).).



**Рис. 3.6.**Запущенная форма программы

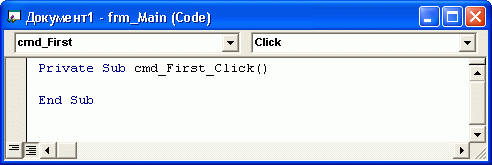
Если вы попробуете нажать на кнопку "Нажми меня!" — ничего не произойдет. Это неудивительно — мы ведь ничего не делали с кнопкой. Пора это исправить. Чтобы снова вернуться в редактор, закройте форму с помощью красной кнопки с крестом, расположенной в правой верхней части ее окна.

Дважды щелкните мышью *по* кнопке cmd\_First. Откроется *редактор кода*. В нем будет автоматически создан каркас процедуры обработчика события Click для кнопки. Об этом читайте дальше.

**3.6. Обработчики событий и подробности о редактировании кода**

Программы на *VBA* управляются событиями. Обработчик — это *подпрограмма*, которая будет выполняться если произойдет соответствующее ей событие.

На [рис. 3.7](https://www.intuit.ru/studies/courses/494/350/lecture/8357?page=3#image.3.7). вы можете видеть окно *редактора кода* с открытым в нем пустым обработчиком.



**Рис. 3.7.**Окно Code с обработчиком события нажатия на кнопку

Обратите внимание на первую строчку кода

Private Sub cmd\_First\_Click()

означает, что эта процедура будет выполнена при щелчке ( Click ) *по* кнопке с именем cmd\_First. Ключевое *слово* *Private* называется модификатором доступности. Оно означает, что данная процедура недоступна для вызова из внешних модулей. Вам не придется настраивать модификаторы доступности вручную — *по* крайней мере, на данном этапе знакомства с *VBA*. Так же вам совсем необязательно запоминать тонкости правильного написания заголовков процедур обработчиков событий. Обычно они создаются автоматически.

У элементов управления существует обработчик события, назначаемый по умолчанию, то есть – при двойном щелчке. Обработчики других событий можно настроить в окне *редактора кода*.

Обратите внимание на верхнюю часть окна *редактора кода*. В левой ее части расположено *поле* со списком, в котором видна надпись cmd\_First. Если щелкнуть *по* кнопке с треугольником, которая расположена рядом с этим полем, откроется *список*, содержащий имена всех элементов управления, расположенных на форме. Также в списке представлена и сама форма, у которой тоже есть свои события, — это строка UserForm. Имя формы в этом списке не применяется, но оно нужно для доступа к форме из внешних программ, например — макросов, обработчиков событий других форм.

В правой части окна расположен *список событий*.

Чтобы создать процедуру обработчика события для элемента управления или формы, выберите нужный элемент управления в левом *поле*, а в правом — название события. Каждый элемент управления имеет уникальный набор событий, однако, в них не слишком сложно ориентироваться. На практике чаще всего используется лишь некоторые из них. Обычно события имеют осмысленные англоязычные названия, *по* которым можно догадаться об их предназначении.

Теперь попытаемся вставить какие-нибудь команды в обработчик события. Пусть это будут строка комментария (она начинается с апострофа), поясняющая работу программы, и *команда* вывода сообщения.

Строки кода внутри программных конструкций принято писать с отступом. Это облегчает восприятие программы. Длинные строки можно разрывать на более короткие, ставя в конце этих строк знак "\_".

Для удобства можно разбивать различные блоки программы пустыми строками.

Иногда при работе в редакторе кода удобно разделить окно на две части. Для этого служит горизонтальная полоска, расположенная в правой части окна, около верхней кнопки полосы прокрутки. Разделив окно, вы можете редактировать один участок кода, сверяясь с другим участком.

В левом нижнем углу окна редактора расположена пара кнопок. *По* умолчанию нажата кнопка Full Module View (Просмотр модуля целиком) то есть в окне редактора отображается код модуля целиком — со всеми процедурами и обработчиками событий. Если вы заняты редактированием какого-то одного модуля и не хотите отвлекаться на остальные — нажмите на кнопку Procedure View (Просмотр процедуры), которая скроет все процедуры кроме процедуры, код которой вы редактируете в настоящий момент.

Для вывода сообщений в *VBA* есть *встроенная команда* MsgBox.

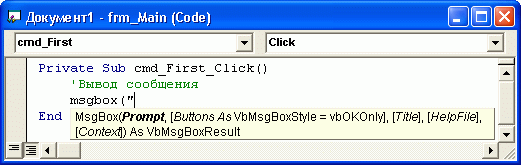
Говоря о *встроенных командах*, часто используют слова "процедуры" и "функции". Главное отличие процедуры от функции – то, что функция возвращает в точку вызова какие-то значения, а процедура — нет. Однако, MsgBox может работать и как функция, и как процедура, поэтому и то и другое наименование будет правомерным.

Как и многие другие команды, она имеет множество параметров. Но вам совсем необязательно помнить их, обращаясь к MsgBox. Дело в том, что *редактор кода* снабжен очень удобной системой подсказок. Так же редактор может автоматически завершать команды ( **Edit • Complete Word** ).

Начните вводить в редакторе текст msgbox( ([рис. 3.8](https://www.intuit.ru/studies/courses/494/350/lecture/8357?page=3#image.3.8).) причем именно строчными буквами. Если в итоге все будет введено верно, редактор автоматически преобразует msgbox в MsgBox. Если этого не произошло — возможно, вы допустили ошибку при написании имени команды. Итак, при правильном вводе команды, редактор распознает ее и предложит *контекстную подсказку*.

В *контекстной подсказке* *по* MsgBox можно почерпнуть информацию о структуре вызова этой функции, о тех параметрах, которые ей можно передать. Параметры — это те данные, с которыми будет работать *функция*. Например, мы передаем MsgBox строку для вывода в окне сообщения. В *контекстной подсказке* *по* MsgBox эта строка называется Prompt (Приглашение). Она выделена жирным шрифтом – это значит, то, что будет сейчас введено, воспримется как Prompt — то есть как текст, который будет выведен в окне сообщения. Текст, который вы передаете функции, должен быть заключен в кавычки.

Если поставить после введенного текста запятую, в *контекстной подсказке* жирным выделится следующий *параметр*. Если вы хотите оставить этот *параметр* в значении *по* умолчанию, ставьте еще одну запятую, и сможете ввести *значение* для очередного параметра и т.д.



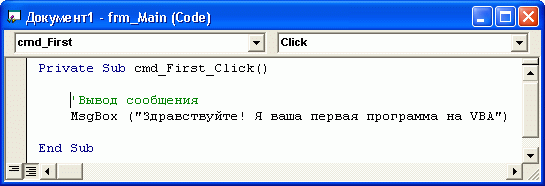
**Рис. 3.8.**Ввод команды в окно редактора

При вызове процедур и функций им можно передавать параметры в таком виде:

MsgBox Prompt:="Привет!"

Здесь мы присвоили параметру *Prompt* *значение* "Привет", то есть наше окно должно вывести сообщение "Привет". Такой подход очень удобно использовать при вызове команд, принимающих множество параметров.

Мы не будем вводить другие параметры для окна сообщения — ограничимся только текстом и завершим ввод. Вот, что у нас получилось ([рис. 3.9](https://www.intuit.ru/studies/courses/494/350/lecture/8357?page=3#image.3.9).).



**Рис. 3.9.**Код обработчика события, выводящий окно сообщения

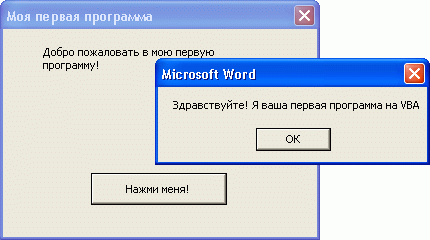
В итоге, код обработчика события будет выглядеть так (Листинг 3.1.):

'Вывод сообщения

MsgBox ("Здравствуйте! Я ваша первая программа на VBA")

Листинг 3.1. Код для вывода сообщения

Запустим форму на выполнение и нажмем на кнопку — появится окно с сообщением ([рис. 3.10](https://www.intuit.ru/studies/courses/494/350/lecture/8357?page=3#image.3.10).).



**Рис. 3.10.**Сообщение, появляющееся при нажатии на кнопку

Теперь поговорим об использовании свойств элементов управления в программном коде.

**3.7. Свойства и методы элементов управления в программах**

[03-02-Свойства элемента управления.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/3/files/03_02.docm) — пример к п 3.7.

Вы уже знаете, что настраивать *элементы управления* можно вручную изменяя их свойства в окне Properties. А что если поменять свойство прямо в программном коде? Давайте попробуем.

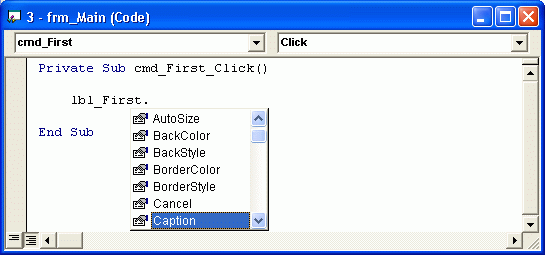
Вернемся к нашему примеру. Модифицируем код обработчика события Click для кнопки cmd\_First таким образом, чтобы при нажатии на эту кнопку изменялся бы текст, который выводит надпись lbl\_First.

Текст надписи задается при помощи свойства Caption. Выше мы уже обсуждали теорию работы с объектами, а теперь пришло время практики.

Чтобы изменить свойство Caption надписи из кода, используется такая конструкция:

lbl\_First.Caption = "Текст"

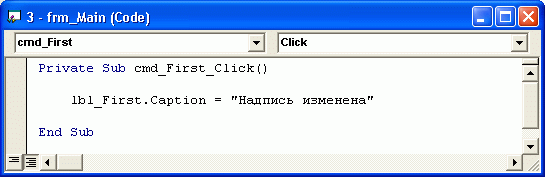
Введите в обработчик события нажатия на кнопку имя элемента управления lbl\_First. Поставьте после него точку. Если вы набрали имя правильно, редактор предложит *список* свойств и методов, доступных данному элементу управления ([рис. 3.11](https://www.intuit.ru/studies/courses/494/350/lecture/8357?page=4#image.3.11).).



**Рис. 3.11.**Подсказка о свойствах и методах объекта в редакторе кода

Вы можете ввести первые буквы нужного свойства или метода, редактор автоматически найдет его в списке. Для окончания ввода вам будет достаточно нажать клавишу Enter. Так же вы можете, воспользовавшись полосой прокрутки, найти свойство или метод вручную и, сделав *по* его наименованию *двойной щелчок* (или, опять же, нажав Enter ) завершить ввод.

После того, как lbl\_First.*Caption* введено, наберите знак = и, в кавычках — текст, который должен быть присвоен параметру Caption. В результате окно *редактора кода* должно выглядеть так ([рис. 3.12](https://www.intuit.ru/studies/courses/494/350/lecture/8357?page=4#image.3.12).)



**Рис. 3.12.**Присвоение нового значения свойству элемента управления

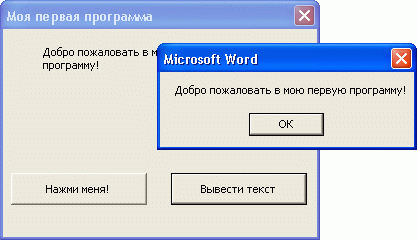
lbl\_First.*Caption* = "Надпись изменена"

Теперь если вы запустите форму и нажмете на кнопку, текст надписи будет изменен.

Аналогично, мы можем считывать значения из свойств элементов управления и использовать их в программах. Например, напишем код, выводящий текст, который записан в свойстве Caption элемента управления lbl\_First, в окне сообщения. Для этого добавим на форму еще одну кнопку, назовем ее cmd\_Second и добавим в ее обработчик события Click такой код:

MsgBox (lbl\_First.Caption)

Чтобы понять этот код, достаточно попытаться "перевести" его на обычный человеческий язык. Получится примерно следующее: "Отобразить в окне сообщения текст, который содержится в свойстве Caption объекта lbl\_First ". Или, еще проще: "Отобразить в окне сообщения то, что написано на элементе управления lbl\_First ". Вот, как выглядит результат работы ([рис. 3.13](https://www.intuit.ru/studies/courses/494/350/lecture/8357?page=4#image.3.13).).



**Рис. 3.13.**Чтение значений из свойств элементов управления

*Элементы управления* имеют свойства *по* умолчанию, обращение к которым возможно и без уточняющего оператора. Например, у надписи это свойство Caption. Значит, два нижеприведенных фрагмента кода для надписи lbl\_First приводят к идентичным результатам:

lbl\_First.Caption = "Новая надпись"

lbl\_First = "Новая надпись"

**3.8. Выводы**

В этой лекции мы познакомились с *VBA*-редактором и основными приемами работы с элементами управления. В следующей мы обсудим некоторые подробности, касающиеся использования других элементов управления, рассмотрим приемы работы с ними.

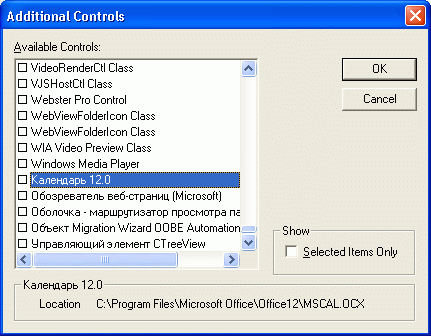
Лекция 4. Подробности об элементах управления

**4.1. Панель инструментов**

Выше мы говорили лишь о кнопке и надписи. Но *VBA* позволяет работать с огромнейшим количеством элементов управления.

На панели инструментов **Toolbox** в редакторе *VBA* отображается лишь их малая часть. Для того чтобы просмотреть установленные в системе элементы и вывести кнопки для их вызова на *панель инструментов*, щелкните правой кнопкой мыши *по* панели и в появившемся *меню* выберите *пункт* **Additional Control** (Дополнительные *элементы управления*).

Даже если вы специально не устанавливали пакеты элементов – вы увидите большой *список* ([рис. 4.1](https://www.intuit.ru/studies/courses/494/350/lecture/8358?page=1#image.4.1).). Мы нашли в этом окне **Календарь 12.0**. Выделим его (для этого надо включить флажок напротив названия) и нажмем **OK** — на панели инструментов *VBA* появится новый элемент управления.



**Рис. 4.1.**Просмотр дополнительных элементов управления

Теперь рассмотрим наиболее часто используемые *элементы управления* и особенности их применения.

**4.2. Form**

[04-01-Формы.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/4/files/04_01.docm) — пример к п. 4.2.

Вы уже знакомы с формами. Они содержат в себе другие *элементы управления*.

Среди событий форм можно отметить следующие.

Initialize (*Инициализация*) — возникает перед появлением формы. На этом этапе форма готовится к открытию и в обработчик этого события обычно включают *операции* для настройки элементов управления, открытия внешних файлов.

Error (Ошибка) — возникает при появлении ошибки в форме.

Terminate (Завершение) — возникает при обычном завершении работы формы. В обработчик этого события можно добавить команды записи протоколов работы приложений, удаления временных файлов, в которых нуждалось *приложение*.

Если вы сомневаетесь в том, когда именно происходит то или иное событие — проведите небольшой эксперимент. Создайте обработчик этого события и добавьте в него функцию MsgBox с какой-нибудь характерной фразой. Когда произойдет событие, вызывающее данный обработчик, появится окно сообщения и вы точно будете знать, на что именно реагирует обработчик.

Если в вашем проекте имеется несколько форм, вам понадобится работать с ними, показывая и скрывая их при необходимости. Чтобы отобразить форму, воспользуйтесь методом Show (Показать). Для скрытия формы используйте метод Hide (Скрыть).

Выполним небольшое упражнение. Создадим несколько форм и настроим навигацию между ними. Одна из форм будет служить главной — из нее вызываются остальные формы.

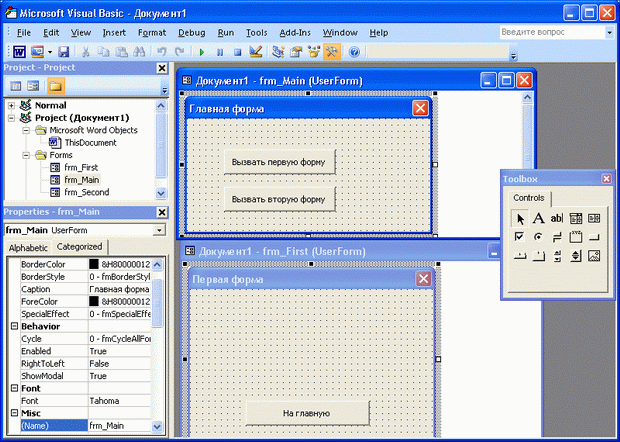
Создайте новый документ Microsoft *Word*, откройте *редактор кода* и добавьте в проект три формы. Настройте их свойства следующим образом ([табл. 4.1](https://www.intuit.ru/studies/courses/494/350/lecture/8358?page=1#table.4.1).).

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица 4.1. Свойства форм | | | |
| **Свойство** | **Форма №1** | **Форма №2** | **Форма №3** |
| Name | frm\_Main | frm\_First | frm\_Second |
| Caption | Главная форма | Первая форма | Вторая форма |

Добавьте на форму frm\_Main две кнопки. Одну из них назовите cmd\_Call\_F1 и подпишите ее "Вызвать первую форму", вторую – cmd\_Call\_F2 с подписью "Вызвать вторую форму". Эти кнопки должны скрывать форму frm\_Main, и, соответственно, вызывать frm\_First и frm\_Second.

На форму frm\_First добавьте кнопку cmd\_First с надписью "На главную". Нажатие на эту кнопку должно скрывать frm\_First и отображать frm\_Main. Аналогичную кнопку (только с именем cmd\_Second ) добавьте на frm\_Second.

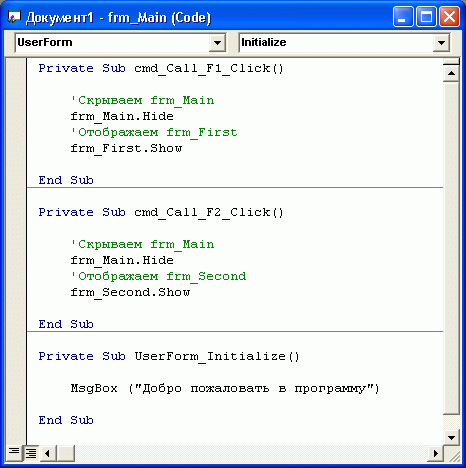
Вот как ([рис. 4.2](https://www.intuit.ru/studies/courses/494/350/lecture/8358?page=1#image.4.2).) выглядит окно проекта.

[](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/4/files/04_02.gif)

[увеличить изображение](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/4/files/04_02.gif)  
**Рис. 4.2.**Формы в окне проекта

Создадим обработчик события Initialize для формы frm\_Main. Добавим в него команду вывода окна сообщения с надписью "Добро пожаловать в программу.

На [рис. 4.3](https://www.intuit.ru/studies/courses/494/350/lecture/8358?page=1#image.4.3). вы можете видеть окно кода формы frm\_Main после создания необходимых обработчиков событий.



**Рис. 4.3.**Обработчики событий в форме frm\_Main

Обработчик события Click для кнопки cmd\_First формы frm\_First выглядит так ([листинг 4.1](https://www.intuit.ru/studies/courses/494/350/lecture/8358?page=1#example.4.1), комментарии опущены).

frm\_First.Hide

frm\_Main.Show

Листинг 4.1. Обработчик Click для cmd\_First

Обработчик события Click кнопки cmd\_Second формы frm\_Second выглядит аналогично — главное отличие — он скрывает frm\_Second

Поэкспериментируйте с этой программой, попробуйте назначить формам другие обработчики событий, выводящие сообщения и посмотрите, что из этого выйдет. Подобная схема работы с формами используется в реальных проектах — тогда, когда нужно организовать взаимодействие пользователя с несколькими формами.

**4.3. Label**

Элемент управления label (*Метка*) используется лишь для вывода подписей к другим элементам управления. Обычно ему не назначают никаких обработчиков событий

**4.4. CommandButton**

Элемент управления CommandButton (Кнопка) обычно используют для выполнения каких-либо команд. Основное событие кнопки — Click (Щелчок). Оно генерируется при щелчке *по* кнопке.

Кнопка может воспринимать и другие события — всего их больше десятка. Обычно об их предназначении можно узнать из их же названий. Например, очевидно, что название события MouseMove переводится как "Перемещение указателя мыши". Это событие будет генерироваться всякий раз, когда над кнопкой будет перемещаться *указатель* мыши. Ниже мы напишем программу, работа которой будет основана на этом событии.

**4.5. TextBox**

[04-02-Текстовое поле.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/4/files/04_02.docm) — пример к п. 4.5.

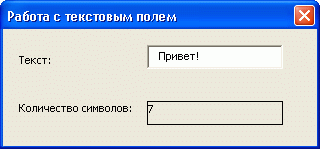
TextBox — текстовое *поле*. Обычно используется для ввода данных пользователем. Текст, который введен в текстовое *поле*, можно получить или изменить, воспользовавшись его свойством Text. Текстовым полям нечасто назначают события, однако, например, для проверки введенных данных сразу после ввода, можно использовать событие Change (Изменение). Оно генерируется всякий раз, когда содержимое поля меняется.

Рассмотрим пример. Создадим форму и добавим на нее два элемента управления — текстовое *поле* txt\_First и надпись lbl\_First. Установим у lbl\_First свойство BorderStyle в fmBorderStyleSingle — благодаря этому вокруг надписи будет отображаться граница. Можете так же добавить пару пояснительных надписей к txt\_First и lbl\_First.

Сделаем *двойной щелчок* *по* текстовому полю. Откроется окно редактора кода с открытым обработчиком события Change. Добавим в обработчик такой код:

lbl\_First.Caption = txt\_First.TextLength

Вам знакомы почти все элементы этого кода кроме свойства текстового поля TextLength. Это свойство позволяет узнать, сколько символов содержится в *поле*. Запустите программу. Попытайтесь ввести что-нибудь в *поле* с клавиатуры — *по* мере ввода символов в текстовое *поле*, их количество будет отображаться на lbl\_First ([рис. 4.4](https://www.intuit.ru/studies/courses/494/350/lecture/8358?page=2#image.4.4).).



**Рис. 4.4.**Работа с текстовым полем

**4.6. ComboBox**

[04-03-Поле со списком.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/4/files/04_03.docm) — пример к п. 4.6.

ComboBox — *поле* со списком. Используется для хранения списков значений. *Поле* следует заполнить перед использованием. Для добавления новых значений используется метод AddItem (Добавить элемент). Обычно вызовы этого метода помещают в обработчик события Initialize формы, на которой расположено *поле*.

Для очистки поля можно использовать метод Clear (Очистить).

После того, как *пользователь* выбрал один из параметров поля, считать выбранное *значение* можно, воспользовавшись свойством Value.

После того, как *пользователь* сделал выбор из поля со списком, выбранное *значение* обрабатывается *по* нажатию какой-либо кнопки. Можно также воспользоваться событием Change (Изменение).

Рассмотрим пример использования элемента управления ComboBox. Заполним его списком фамилий, а при выборе фамилии из списка, будем выводить ее в окне сообщения.

Создадим форму frm\_Main, расположим на ней *поле* со списком, назовем его cbo\_First. Теперь создадим обработчик события Initialize для frm\_Main и добавим туда такой код ([листинг 4.2](https://www.intuit.ru/studies/courses/494/350/lecture/8358?page=2#example.4.2)):

cbo\_First.AddItem ("Иванов")

cbo\_First.AddItem ("Петров")

cbo\_First.AddItem ("Сидоров")

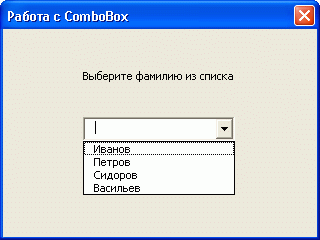
cbo\_First.AddItem ("Васильев")

Листинг 4.2. Код обработчика события Initialize для frm\_Main

Создадим обработчик события Change для cbo\_First и добавим в него команду вывода сообщения:

MsgBox (cbo\_First.Value)

Теперь запустим форму ([рис. 4.5](https://www.intuit.ru/studies/courses/494/350/lecture/8358?page=2#image.4.5).). При нажатии на кнопку с треугольником, *поле* раскрывается, при выборе одной из строк поля, ее *значение* отображается в окне сообщения.



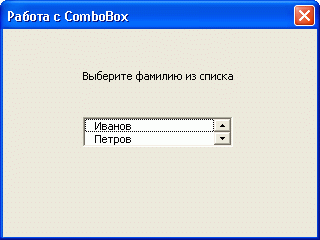
**Рис. 4.5.**Работа с ComboBox

**4.7. ListBox**

[04-04-Cписок.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/4/files/04_04.docm) — пример к п. 4.7.

ListBox — *список*. Обычно используется для представления списков данных.

Работа с ListBox аналогична работе с ComboBox. На [рис. 4.6](https://www.intuit.ru/studies/courses/494/350/lecture/8358?page=2#image.4.6). вы можете видеть форму с расположенным на ней списком, реализующую ту же функциональность, что и в примере к полю со списком.



**Рис. 4.6.**Работа с ListBox

**4.8. CheckBox**

[04-05-Флажок.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/4/files/04_05.docm)— пример к п. 4.8.

CheckBox — флажок. Используется для включения и отключения каких-либо опций. Элемент управления представляет собой *поле*, где можно устанавливать и снимать флажок и надпись, где обычно выводится название и назначение флажка.

Если флажок установлен — его свойство Value (*Значение*) устанавливается в True (*Истина*), если не установлен — в False (*Ложь*). Для того чтобы программно установить или снять флажок, можно воспользоваться его свойством Value, приравняв ему True или False, соответственно.

Еще одно важное свойство флажка — TripleState. Если оно включено — флажок помимо True или False может иметь *значение* Null, которое можно интерпретировать как "Пустой флажок". *Null*-флажок нельзя модифицировать — он закрашен серым цветом.

Сами по себе, без использования специальных конструкций языка, флажки приносят мало пользы. Эффективно работать с ними можно, используя операторы принятия решений.

С флажком можно сопоставить событие **Change** (Изменить). Оно выполняется всякий раз при установке или снятии флажка.

Давайте рассмотрим пример работы с флажками. Создадим форму frm\_First и разместим на ней три флажка – chk\_1, chk\_2 и chk\_3. Добавим на форму пару кнопок – cmd\_SetAll с надписью "Установить все флажки" и cmd\_ClearAll с надписью "Снять все флажки". Думаю, вы видели что-то подобное во многих программах — часто пользователю предоставляется возможность не кликать *по* множеству флажков вручную, устанавливая или снимая их, а сделать эту работу одним нажатием специальной кнопки. Точно так же, можно создать кнопку, которая устанавливает определенный набор флажков из всех, реализуя тем самым какую-нибудь особенную настройку программы.

Добавим обработчик события Click для cmd\_SetAll ([листинг 4.3](https://www.intuit.ru/studies/courses/494/350/lecture/8358?page=2#example.4.3)). Если попытаться выразить обычным языком то, что должно произойти *по* нажатию этой кнопки, то получится следующее: "Установить все три флажка, то есть, сделать их *параметр* Value равным *True*".

chk\_1.Value = True

chk\_2.Value = True

chk\_3.Value = True

Листинг 4.3. Обработчик события Click для cmd\_SetAll

Рассуждая аналогично, пишем код для события Click кнопки cmd\_ClearAll ([листинг 4.4](https://www.intuit.ru/studies/courses/494/350/lecture/8358?page=2#example.4.4)). Очевидно, что единственное его отличие от предыдущего кода заключается в смене True на False.

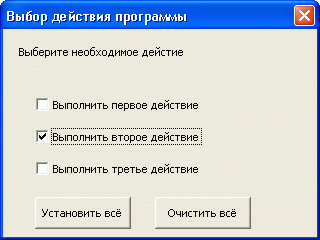
chk\_1.Value = False

chk\_2.Value = False

chk\_3.Value = False

Листинг 4.4. Обработчик события Click для cmd\_ClearAll

На [рис. 4.7](https://www.intuit.ru/studies/courses/494/350/lecture/8358?page=2#image.4.7). вы можете видеть запущенную форму.



**Рис. 4.7.**Интерфейс с использованием CheckBox

**4.9. OptionButton**

[04-06-Рамки и радиокнопки.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/4/files/04_06.docm) — пример к пп. 4.9. – 4.10.

OptionButton — радиокнопка. Не удивляйтесь такому названию. Оно отражает особенности этого элемента управления. Радиокнопки обычно используются группами, а в группе может быть выбрана лишь одна кнопка. Это напоминает выбор одной радиостанции с помощью одной из кнопок на радиоприемнике.

Основное свойство радиокнопки — это Value (*Значение*) — если оно равно False (*Ложь*) — радиокнопка не выбрана, если True (*Истина*) — выбрана. Свойство TripleState позволяет присваивать элементу управления *параметр* Null, который, как в случае с флажками, блокирует элемент.

Все радиокнопки, расположенные на одной форме, автоматически объединяются в одну группу. Поэтому из всех этих кнопок выбрана может быть лишь одна. Как же быть, если на форме надо расположить несколько групп таких кнопок? Ответ на этот вопрос кроется в использовании так называемых фреймов или рамок (элемент управления Frame ). Радиокнопки, размещенные в пределах рамки, обрабатываются отдельно от кнопок на других рамках или на формах.

**4.10. Frame**

Frame — рамка. Рамки обычно используют для группировки объектов. Например, на форме может быть расположено несколько групп элементов управления, выполняющих различные функции. Логично будет сгруппировать их с помощью рамок. Для этого нужно создать на форме рамку и перетащить на нее остальные *элементы управления*. Как правило, рамкам не назначают обработчики событий, используя их как контейнеры для других объектов.

Если вы хотите обратиться к элементу управления, который расположен на рамке, используйте такой же *синтаксис*, как и для доступа к объекту, расположенному на форме.

Давайте рассмотрим пример совместного использования рамок и радиокнопок. Предположим, нам нужно спроектировать *интерфейс* для настройки сохранения файлов из программы. В *интерфейс* должны входить три группы параметров – для выбора формата сохраняемого файла, для управления именованием файла (вручную или автоматически), и местом сохранения (так же вручную или автоматически).

Создадим форму frm\_First и добавим на нее две рамки — fra\_First и fra\_Second. Добавим две радиокнопки на форму (назовем их opt\_1 и opt\_2 ). Так же добавим *по* две радиокнопки в каждую из рамок. В рамку fra\_First добавим кнопки opt\_3 и opt\_4, а в fra\_Second — opt\_5 и opt\_6. Установим *параметр* TripleState равным True для всех радиокнопок. Подпишем радиокнопки и рамки так, как указано в [табл. 4.2](https://www.intuit.ru/studies/courses/494/350/lecture/8358?page=3#table.4.2).

|  |  |  |
| --- | --- | --- |
| Таблица 4.2. Подписи рамок и радиокнопок | | |
| **Имя элемента управления** | **Свойство Caption** | **Value** |
| Fra\_First | Имена файлов |  |
| Fra\_Second | Место сохранения файлов |  |
| opt\_1 | Сохранить в TXT-файл | True |
| opt\_2 | Сохранить в DOCX-файл | False |
| opt\_3 | Автоматически | True |
| opt\_4 | Задать вручную | False |
| opt\_5 | Автоматически | True |
| opt\_6 | Задать вручную | False |

Добавим на форму кнопку, назовем ее **cmd\_SetAuto** и подпишем: "Автоматически", добавим еще одну кнопку – **cmd\_SetManual** и подпишем ее "Вручную". Теперь создадим обработчики событий **Click** для кнопок.

Кнопка cmd\_SetAuto устанавливает радиокнопки с надписью "Автоматически" ([листинг 4.5](https://www.intuit.ru/studies/courses/494/350/lecture/8358?page=3#example.4.5))

opt\_3.Value = True

opt\_5.Value = True

Листинг 4.5. Обработчик события Click для cmd\_SetAuto

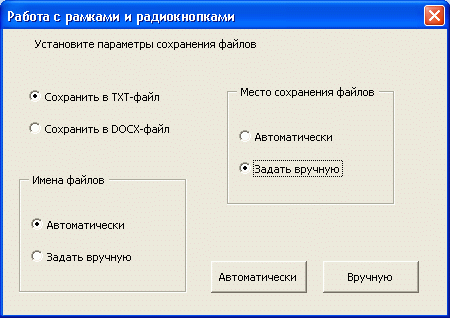
Кнопка cmd\_SetManual устанавливает радиокнопки с надписью "Вручную" ([листинг 4.6](https://www.intuit.ru/studies/courses/494/350/lecture/8358?page=3#example.4.6))

opt\_4.Value = True

opt\_6.Value = True

Листинг 4.6. Обработчик события Click для cmd\_SetManual

На [рис. 4.8](https://www.intuit.ru/studies/courses/494/350/lecture/8358?page=3#image.4.8). вы можете видеть форму этого примера. Поэкспериментируйте с ней — убедитесь, что группы радиокнопок, расположенных в различных рамках, действуют независимо друг от друга.



**Рис. 4.8.**Работа с рамками и радиокнопками

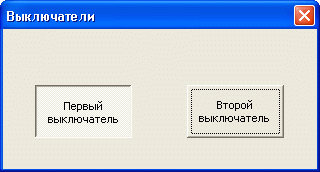
**4.11. ToggleButton**

ToggleButton — выключатель. Это кнопка, которая может существовать в двух состояниях — включенном и выключенном. *По* действию она напоминает флажок, и обычно используется точно так же.

Если кнопка находится во включенном состоянии — это может означать включение какой-либо опции, если в выключенном — выключение. Основное свойство такой кнопки — Value (*Значение*). Оно установлено в False (*Ложь*), когда кнопка находится в обычном состоянии, и в True (*Истина*), когда кнопка нажата.

При изменении состояния кнопки генерируется событие Change (Изменение) — его можно использовать для проверки установленного значения и проведения каких-либо действий.

На [рис. 4.9](https://www.intuit.ru/studies/courses/494/350/lecture/8358?page=3#image.4.9). вы можете видеть форму с размещенными на ней выключателями. Один из них включен, другой – выключен.



**Рис. 4.9.**Выключатели на форме

**4.12. ScrollBar**

[04-07-Полоса прокрутки.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/4/files/04_07.docm) — пример к п. 4.12.

ScrollBar — *полоса прокрутки*. Используется для плавного изменения каких-либо значений. Например, это может быть выбор параметра из диапазона значений. При настройке этого элемента управления следует учесть, что для его нормальной работы надо установить следующие свойства:

* Max (Максимум) — максимальное значение полосы.
* Min (Минимум) — минимальное значение полосы
* Small Change (Маленькое изменение) — шаг, с которым изменяется значение полосы при прокрутке с помощью кнопок.
* Large Change (Большое изменение) — шаг, с которым меняется значение при перетаскивании бегунка полосы с помощью мыши.
* Узнать текущее значение полосы можно, воспользовавшись ее свойством Value (Значение).

При изменении значения полосы прокрутки генерируется событие Change (Изменение). Оно происходит после каждого изменения значения полосы. Если вы перетаскиваете бегунок с помощью мыши, новое *значение* полосы прокрутки будет сгенерировано лишь тогда, когда вы отпустите кнопку мыши. Для того чтобы непрерывно изменять *значение* полосы вслед за перетаскиванием бегунка вручную, воспользуйтесь обработчиком события Scroll (Прокрутка).

Расположение кнопок (горизонтальное или вертикальное) выбирается автоматически в зависимости от ширины и высоты полосы прокрутки.

Давайте рассмотрим пример, иллюстрирующий возможности элемента управления ScrollBar и разницу между событиями Change и Scroll. Создадим форму, добавим на нее полосу прокрутки с именем scr\_First, оставим ее параметры равными *по* умолчанию, и две надписи – lbl\_Change и lbl\_Scroll. Они будут содержать информацию о значении полосы прокрутки, но lbl\_Change будет обновляться *по* событию полосы прокрутки Change ([листинг 4.7](https://www.intuit.ru/studies/courses/494/350/lecture/8358?page=3#example.4.7)), а lbl\_Scroll — *по* событию Scroll ([листинг 4.8](https://www.intuit.ru/studies/courses/494/350/lecture/8358?page=3#example.4.8)).

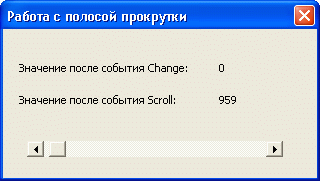
lbl\_Change.Caption = scr\_First.Value

Листинг 4.7. Обработчик события Change для scr\_First

lbl\_Scroll.Caption = scr\_First.Value

Листинг 4.8. Обработчик события Scroll для scr\_First

На [рис. 4.10](https://www.intuit.ru/studies/courses/494/350/lecture/8358?page=3#image.4.10). вы можете видеть форму нашего примера в момент перетаскивания бегунка вручную.



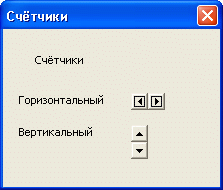
**Рис. 4.10.**Работа с полосой прокрутки

До тех пор, пока мы не отпустили левую кнопку мыши, при каждом перемещении бегунка вручную будет генерироваться событие Scroll — *значение* в надписи lbl\_Scroll будет меняться при перемещении бегунка. А вот *значение* в *поле* lbl\_Change, которое обновляется *по* событию Change, изменится лишь тогда, когда мы отпустим кнопку мыши. В то же время, изменяя *значение* полосы прокрутки с помощью кнопок прокрутки, мы можем наблюдать лишь изменения поля lbl\_Change так как нажатия на эти кнопки генерируют событие Change. А вот *поле* lbl\_Scroll при использовании кнопок обновляться не будет – событие Scroll не генерируется.

**4.13. SpinButton**

SpinButton — *счетчик*. Этот элемент управления напоминает полосу прокрутки без центрального бегунка. Он служит для выбора значений из небольшого диапазона. Работа с ним аналогична работе с полосой прокрутки. Главное отличие — отсутствие у счетчика события Scroll (Прокрутка) и свойства Large Change (Большое изменение).

На [рис. 4.11](https://www.intuit.ru/studies/courses/494/350/lecture/8358?page=4#image.4.11). вы можете видеть форму с парой счетчиков – горизонтальным и вертикальным. Как и в случае с полосой прокрутки, расположение кнопок на элементе управления выбирается автоматически в зависимости от ширины и высоты.



**Рис. 4.11.**Счетчики на форме

**4.14. Image**

[04-08-Изображения.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/4/files/04_08.docm) — пример к п. 4.14., файлы 1.jpg и 2.jpg содержат графический материал к этому примеру.

Image — изображение. Этот элемент управления используют для вывода на форму изображений.

За картинку, которая будет отображаться в Image, отвечает *параметр* Picture (Изображение). Настраивая этот *параметр* в панели Properties, вы должны будете выбрать нужный графический *файл* с помощью стандартного окна поиска файлов.

Для того, чтобы изменить картинку в ходе выполнения программы, вы должны воспользоваться специальной функцией LoadPicture таким образом:

Имя\_рисунка.picture = LoadPicture (полный\_путь\_к\_файлу)

Среди других важных свойств рисунка можно отметить такие, как *AutoSize* (Автоматический размер). Если свойство установлено в True (*Истина*) – элемент управления автоматически изменяет свой размер в соответствии с размером загруженного в него изображения.

Рисунку может быть назначено какое-нибудь событие. Например, при щелчке (событие Click ), который представляет собой фотографию сотрудника, может выводиться краткая *информация* о нем и т.д.

Рассмотрим пример, в котором изображение в элементе управления Image меняется при нажатии на кнопки. Для этого примера нам понадобится пара графических файлов. Мы заранее создали их – каждый из них имеет размер 100х100 пикселей и формат jpg.

Создадим новый документ Microsoft *Word* и сохраним его в какой-нибудь папке. После этого нужно скопировать в ту же папку файлы изображений. Вы можете использовать в своих проектах файлы, расположенные произвольно, но мы поступим именно так. Теперь создадим форму и добавим на нее картинку с именем img\_First. С помощью свойства Picture (Изображение) выберем изображение, которое будет отображено на элементе управления. Настроим размер элемента управления таким образом, чтобы изображение поместилось в нем полностью.

Теперь добавим на форму две кнопки — cmd\_First и cmd\_Second. При нажатии на кнопке cmd\_First в img\_First должна отображаться картинка 1.jpg, ([листинг 4.9](https://www.intuit.ru/studies/courses/494/350/lecture/8358?page=4#example.4.9)) при нажатии кнопки cmd\_Second — 2.jpg ([листинг 4.10](https://www.intuit.ru/studies/courses/494/350/lecture/8358?page=4#example.4.10)). Так как картинки расположены в той же папке, что и наш *файл*, мы можем использовать для их загрузки *относительный путь*, то есть использовать лишь *имя файла*. Если же ваши картинки расположены где-нибудь еще — надежнее всего будет указать абсолютный *путь* к ним (например, похожий на этот: "C:\Images\1.jpg" ). Для изменения свойства Picture элемента управления img\_First в коде программы нужно воспользоваться уже упомянутой функцией LoadPicture.

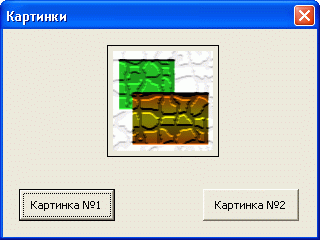
img\_First.Picture = LoadPicture("1.jpg")

Листинг 4.9. Обработчик события Click для cmd\_First

img\_First.Picture = LoadPicture("2.jpg")

Листинг 4.10. Обработчик события Click для cmd\_Second

На [рис. 4.12](https://www.intuit.ru/studies/courses/494/350/lecture/8358?page=4#image.4.12). вы можете видеть форму нашего примера.



**Рис. 4.12.**Работа с элементом управления Image

**4.15. MultiPage**

[04-09-Набор страниц.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/4/files/04_09.docm) — пример к п. 4.15.

MultiPage — набор страниц. С помощью этого элемента можно создать многостраничное окно. *Доступ* к отдельным частям такого окна осуществляется посредством выбора вкладок.

Управлять вкладками можно с помощью контекстного *меню*, которое появляется при щелчке *по* элементу управления. Вкладки организованы в коллекцию Pages (Страницы). Чтобы обратиться к элементу управления, расположенному на одной из вкладок элемента управления MultiPagе, нужно использовать такую конструкцию:

Имя\_элемента\_MultiPage.Имя\_вкладки.Имя\_элемента\_на\_вкладке

Важнейшие свойства объекта MultiPage следующие:

* Value (Значение) — возвращает номер активной вкладки (нумерация начинается с нуля).
* SelectedItem — возвращает выбранную страницу.

При выборе вкладки генерируется событие Change.

Выполним пример. Создадим форму, поместим на нее элемент управления MultiPage, назовем его mlt\_First. Добавим одну вкладку к набору вкладок, добавляемому на элемент *по* умолчанию. Вкладки имеют имена Page1, Page2 и т.д. Чтобы настраивать свойства каждой вкладки *по* отдельности, будем выделять эти вкладки в окне редактора. В [табл. 4.3](https://www.intuit.ru/studies/courses/494/350/lecture/8358?page=4#table.4.3). приведены параметры вкладок и *список* элементов управления на каждой из них.

|  |  |  |
| --- | --- | --- |
| Таблица 4.3. Параметры вкладок | | |
| **Имя вкладки** | **Надпись** | **Элементы управления на вкладке** |
| p\_1 | Вкладка №1 | txt\_First — текстовое поле; cmd\_First — кнопка с надписью "Перенести текст на вкладку №3" |
| p\_2 | Вкладка №2 |  |
| p\_3 | Вкладка №3 | Lbl\_First — надпись |

Настроим событие Change для mlt\_First таким образом, чтобы при выборе каждой вкладки выводилось бы окно сообщения с именем этой вкладки ([листинг 4.11](https://www.intuit.ru/studies/courses/494/350/lecture/8358?page=4#example.4.11))

MsgBox (mlt\_First.SelectedItem.Name)

Листинг 4.11. Обработчик события Change для mlt\_First

Теперь поработаем с событием Click кнопки cmd\_First, которая расположена на вкладке p\_1. *По* нажатию на эту кнопку текст, который введен в текстовое *поле* txt\_First, должен быть перенесен на надпись lbl\_First, которая расположена на вкладке p\_3. Этот обработчик проиллюстрирует работу с элементами управления, расположенными на различных вкладках ([листинг 4.12](https://www.intuit.ru/studies/courses/494/350/lecture/8358?page=4#example.4.12)).

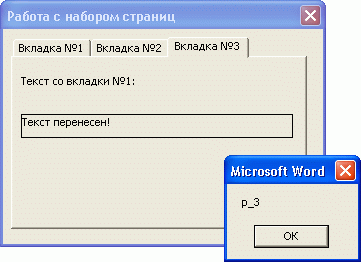
mlt\_First.Pages.Item(2).lbl\_First.Caption = \_

mlt\_First.Pages.Item(0).txt\_First.Text

Листинг 4.12. Обработчик события Click для cmd\_First

В данном случае Item(0) представляет первую страницу в коллекции Pages (вместо индекса 0 можно использовать имя вкладки), Item(2) — третью. Обратите внимание на то, что при вводе части строки после Item(2) и Item(0) перестает работать помощник *по* синтаксису. Однако после ввода данных они проверяются на правильность.

На [рис. 4.13](https://www.intuit.ru/studies/courses/494/350/lecture/8358?page=4#image.4.13). вы можете видеть форму после выбора одной из страниц.



**Рис. 4.13.**Работа с набором страниц

**4.16. TabStrip**

[04-10-Набор вкладок.xlsm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/4/files/04_10.xlsm)— пример к п. 4.16.

TabStrip — набор вкладок. Используется для организации нескольких рабочих областей на форме. В *объект* входит семейство Tabs (Вкладки), которое содержит все активные вкладки.

Основное событие объекта — Change (Изменение). Оно происходит при смене вкладки.

Наборы вкладок используются достаточно редко. Например, их можно применить, чтобы *по*-разному заполнять одни и те же поля, *по*-разному устанавливать флажки и т.д.

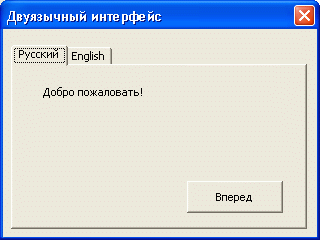
Дело в том, что TabStrip не содержит *элементы управления* внутри себя. Он служит как бы "фоном" для размещения элементов.

Cамый яркий пример использования TabStrip — это создание приложений с многоязыковым интерфейсом. Рассмотрим пример.

Добавим в проект форму frm\_First, разместим на ней элемент управления TabStrip с именем tab\_First. *По* умолчанию у TabStrip две вкладки. Первая *по* порядку вкладка имеет номер 0, вторая — 1 и т.д. Для того, чтобы узнать номер активной вкладки, можно воспользоваться свойством Value элемента управления TabStrip.

Для того, чтобы изменить надписи на вкладках, добавить новые и т.д. служит контекстное *меню*, которое появляется на элементе управления TabStrip при щелчке *по* нему правой кнопкой мыши. Подпишем первую вкладку Русский, вторую — English.

Добавим на форму, поверх tab\_First, метку с именем lbl\_First и текстом Добро пожаловать, а так же кнопку cmd\_First с надписью Вперед ([рис. 4.14](https://www.intuit.ru/studies/courses/494/350/lecture/8358?page=5#image.4.14).).



**Рис. 4.14.**Набор вкладок на форме

Теперь добавим обработчик события Change для tab\_First. ([листинг 4.13](https://www.intuit.ru/studies/courses/494/350/lecture/8358?page=5#example.4.13)).

If tab\_First.Value = 0 Then

lbl\_First.Caption = "Добро пожаловать!"

cmd\_First.Caption = "Вперед"

End If

If tab\_First.Value = 1 Then

lbl\_First.Caption = "Welcome!"

cmd\_First.Caption = "Next"

End If

Листинг 4.13. Обработчик события Change для tab\_First

В этом листинге вы можете видеть конструкцию сравнения и *принятия решений* If-End If. Подробнее о ней мы будем говорить в ["Основы VBA: синтаксис, переменные, типы данных"](https://www.intuit.ru/studies/courses/494/350/lecture/8360). Если "перевести" один из блоков листинга 4.13. на обычный язык, то получится следующее:

Если tab\_First.Value=0 Тогда

lbl\_First.Caption = "Добро пожаловать!"

cmd\_First.Caption = "Вперед"

Конец сравнения

То есть в блоке If мы сравниваем свойство Value и, если оно равно 0, выполняем код, который находится между If и End If. Иначе мы ничего не выполняем и переходим к следующему сравнению. Одно из них выполнится в любом случае, так как на элементе tab\_First есть лишь две вкладки.

В результате, если выбрана первая вкладка (*индекс* 0), свойствам Caption элементов управления lbl\_First и cmd\_First присваиваются русскоязычные слова. Если же выбрана вкладка с индексом 1, элементам управления присваиваются англоязычные названия.

**4.17. Calendar**

[04-11-Календарь.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/4/files/04_11.docm) — пример к п. 4.17.

*Calendar* — календарь. Этот *объект* удобно использовать для ввода и отображения даты. Для того чтобы считать дату, можно использовать его свойство Value (*Значение*). С помощью этого же свойства можно установить дату на календаре.

При изменении дня недели генерируется событие Click (Щелчок), при смене месяца или года – события NewMonth (Новый месяц) и NewYear (Новый год).

Давайте рассмотрим пример. Добавим на форму элемент управления *Calendar*, назовем его cld\_First, добавим второй календарь с именем cld\_Second. Добавим на форму кнопку — cmd\_First, с подписью "Вывести дату". *По* нажатию на кнопку cmd\_First в окне сообщения должна выводиться дата, установленная в cld\_First ([листинг 4.14](https://www.intuit.ru/studies/courses/494/350/lecture/8358?page=5#example.4.14)).

При изменении даты в календаре cld\_First, в календаре cld\_Second должен устанавливаться тот же месяц, который установлен в cld\_First, число месяца должно устанавливаться на 1 день месяца ([листинг 4.15](https://www.intuit.ru/studies/courses/494/350/lecture/8358?page=5#example.4.15)).

MsgBox (cld\_First.Value)

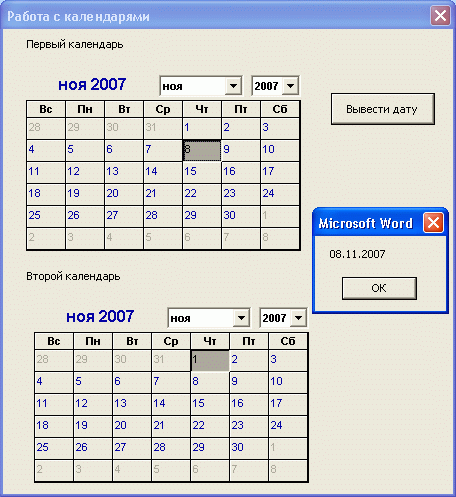
Листинг 4.14. Обработчик события Click для cmd\_First

cld\_Second.Month = cld\_First.Month

cld\_Second.Day = 1

Листинг 4.15. Обработчик события Click для cld\_First

На [рис. 4.15](https://www.intuit.ru/studies/courses/494/350/lecture/8358?page=5#image.4.15). вы можете видеть форму с парой календарей.

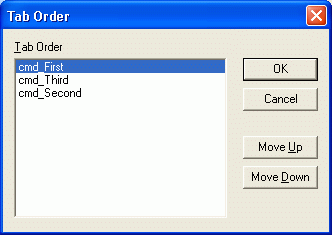


**Рис. 4.15.**Работа с календарем

**4.18. Последовательность обхода элементов управления**

Перемещаться между элементами управления, добавленными на форму, можно, нажимая клавишу **Tab** на клавиатуре. Это — стандартная возможность Windows-программ. *По* умолчанию объекты обходятся в том порядке, в котором они добавляются на форму. Однако, иногда удобно установить свой порядок обхода элементов, назначить какой-то элемент управления активным *по* умолчанию и т.д.

*Элементы управления* имеют свойство TabIndex (*Индекс* табуляции). Этот *индекс* начинается с 0, что соответствует первому элементу управления. Настроить *порядок табуляции* можно, модифицировав этот *параметр*. Однако удобнее воспользоваться специальным окном. Вызвать его можно командой View • Tab Order (Вид • Порядок перехода) ([рис. 4.16](https://www.intuit.ru/studies/courses/494/350/lecture/8358?page=6#image.4.16).). Имена элементов управления расположены в этом окне в порядке обхода. Чтобы изменить этот порядок, воспользуйтесь кнопками Move Up (Переместить вверх) и Move Down (Переместить вниз).



**Рис. 4.16.**Настройка порядка обхода элементов

Для того чтобы перевести фокус ввода на элемент управления программно, воспользуйтесь методом SetFocus (Установить фокус ввода). Вызов этого метода используется без параметров.

Если вы не хотите, чтобы *пользователь* мог переводить фокус ввода на элемент управления с помощью кнопки Tab, установите свойство элемента управления TabStop в False (*Ложь*).

**4.19. Последние штрихи: расположение элементов на форме**

Работая с элементами управления, вы можете столкнуться с проблемой настройки их размеров и взаимного расположения. Эти параметры можно настраивать вручную — перетаскивая элементы *по* форме с помощью мыши или модифицируя их свойства. Но есть еще один способ решения этой проблемы — воспользоваться командами *меню* **Format** (*Форматирование*). Давайте рассмотрим основные пункты этого *меню* и особенности их использования.

**Align** (Выравнивание) — выравнивание элементов относительно различных опорных точек. Например — *команда* **Left** (*По* левому краю) выравнивает все элементы *по* левому краю самого левого элемента.

**Make Same Size** (Сделать одинакового размера) — команды этого подменю позволяют привести выделенные *элементы управления* к одному размеру. Размеры изменяются в большую сторону.

**Size to Fit** (Точный размер) — позволяет изменить размер элемента управления так, чтобы он соответствовал содержимому элемента. Например, кнопка с надписью будет изменена таким образом, чтобы размера этой кнопки в точности хватило бы для размещения на ней надписи.

**Size to Grid** (Размер *по* сетке) — выравнивает размер элемента управления так, чтобы он точно согласовывался с координатными точками, нанесенными на форму.

**Horizontal Spacing** (Горизонтальное *расстояние*) — позволяет настраивать горизонтальное *расстояние* между элементами формы.

**Vertical Spacing** (Вертикальное *расстояние*) — настраивает вертикальное *расстояние* между элементами.

**Center in Form** (Центрировать в форме) — позволяет центрировать элементы в форме *по* горизонтали ( **Horizontal** ) и *по* вертикали ( **Vertical** ).

**Arrange Buttons** (Выровнять кнопки) — выравнивает кнопки *по* правому ( **Right** ) или *по* левому ( **Left** ) краю формы.

**Group** (Группировать) — позволяет объединить несколько выделенных элементов управления. Группу элементов можно, например, перетаскивать, выделив одним щелчком мыши. Группировка элементов не отменяет возможности работать с ними *по* отдельности. Для этого достаточно сначала выделить группу, а потом — нужный элемент управления в группе.

**Ungroup** (Разгруппировать) — разгруппировывает ранее созданную группу элементов.

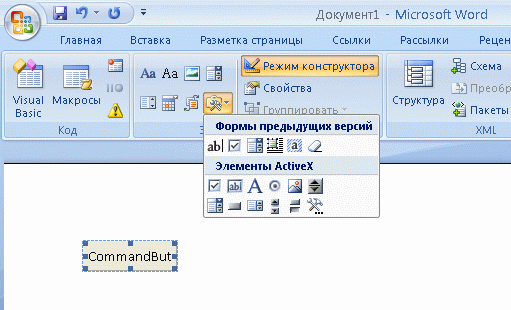
**Order** (Порядок) — команды этого подменю отвечают за порядок расположения элементов на форме. Эти команды имеют смысл тогда, когда элементы перекрываются. *По* умолчанию более поздние *по* времени создания элементы накладываются на более ранние при совпадении их мест расположения. Так, *команда* **Bring to Front** (Переместить на передний план) перемещает элемент на передний план, **Bring to Back** (Переместить на задний план) перемещает элемент на задний план. **Bring Forward** (Переместить вперед) – перемещает элемент на один уровень вперед, **Bring Backward** (Переместить назад) — на один уровень назад.

**4.20. Элементы управления и документы**

[04-12-Кнопка в документе.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/4/files/04_12.docm) — пример к п. 4.20.

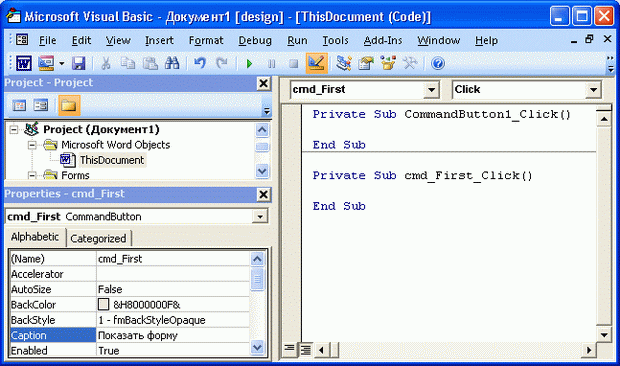
*Элементы управления* можно размещать не только на формах, но и в документах. Чтобы создать элемент управления в документе, нужно переключиться в *режим конструктора*. В Microsoft *Word* и Microsoft *Excel* это можно сделать с помощью кнопки **Режим конструктора**, которая расположена в группе **Элементы управления** вкладки **Разработчик**.

Включите *режим конструктора*, найдите среди элементов управления кнопку ([рис. 4.17](https://www.intuit.ru/studies/courses/494/350/lecture/8358?page=6#image.4.17).) и разместите ее в документе.



**Рис. 4.17.**Добавление элементов управления в документы

Чтобы снабдить кнопку полезной нагрузкой в виде обработчика события или настроить ее свойства сделайте *по* кнопке *двойной щелчок*. Появится уже знакомый вам *редактор кода* ([рис. 4.18](https://www.intuit.ru/studies/courses/494/350/lecture/8358?page=6#image.4.18))

[](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/4/files/04_18.gif)

[увеличить изображение](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/4/files/04_18.gif)  
**Рис. 4.18.**Настройка свойств кнопки

Обратите внимание на то, что код, который соответствует кнопке, расположен в модуле редактируемого документа ( ThisDocument ).

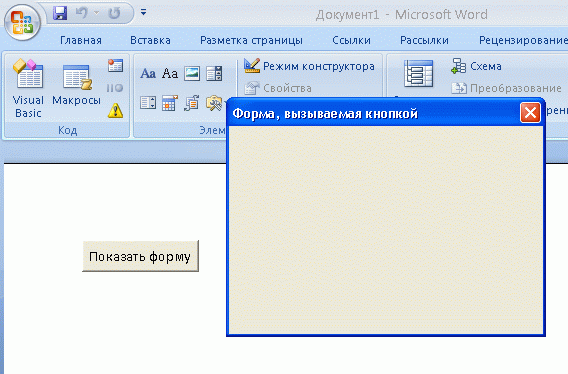
На [рис. 4.18](https://www.intuit.ru/studies/courses/494/350/lecture/8358?page=6#image.4.18). в модуле видны два обработчика события. Дело в том, что мы переименовали кнопку — назвали ее cmd\_First, поэтому нам пришлось создавать новый обработчик так как старый, созданный тогда, когда кнопка называлась CommandButton1, уже неактуален — его можно удалить. Также мы отредактировали свойство кнопки Caption, введя туда текст "Показать форму". Этот текст тут же отобразится на кнопке. Если текст не поместился на кнопке — можно изменить ее размеры.

Добавим в документ форму frm\_First. Чтобы вызвать эту форму с помощью добавленной в документ кнопки, разместим в обработчике ее события Click такой код:

frm\_First.Show

Кнопка в документе заработает, когда вы выключите *режим конструктора*.

На [рис. 4.19](https://www.intuit.ru/studies/courses/494/350/lecture/8358?page=6#image.4.19). вы можете видеть кнопку и запущенную с ее помощью форму.



**Рис. 4.19.**Форма, вызываемая кнопкой

**4.21. Убегающая кнопка**

[04-13-Убегающая кнопка.xlsm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/4/files/04_13.xlsm) — пример к п. 4.21.

После того, как мы разобрались со свойствами и *событиями элементов управления*, предлагаю провести с ними интересный эксперимент. Для этого мы напишем простую, но довольно забавную программу.

**4.21.1. Задание**

1. Создать в Microsoft Excel форму шириной 350 пикселей и высотой 200 пикселей.
2. Разместить на форме кнопки "Да", "Нет" и надпись: "Хотите ли вы получить прибавку к зарплате?".
3. Настроить свойства кнопки с надписью "Да" таким образом, чтобы на нее нельзя было перевести фокус ввода.
4. Написать программу, которая при наведении указателя мыши на кнопку с надписью "Да" перемещала бы эту кнопку в пределах формы.

**4.21.2. Решение**

Создадим новую форму, установим следующим образом ее свойства:

Name (Имя) — frm\_MovingButton

Width (Ширина) — 350

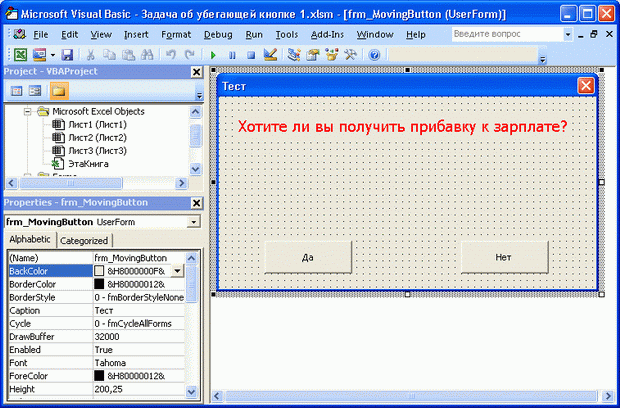
Height (Высота) — 200

Caption (Заголовок формы) — Тест

Добавим на форму следующие элементы управления ([табл. 4.4](https://www.intuit.ru/studies/courses/494/350/lecture/8358?page=7#table.4.4).)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Таблица 4.4. Элементы управления и их свойства | | | | | |
| **Название** | **Name (Имя)** | **Caption (Надпись)** | **Tabstop (Фокус ввода по нажатию клавиши Tab)** | **Height (Высота)** | **Width (Ширина)** |
| Кнопка №1 | cmd\_Yes | Да | False | 30 | 80 |
| Кнопка №2 | cmd\_No | Нет |  | 30 | 80 |
| Надпись | lbl\_text | Хотите ли вы получить прибавку к зарплате? |  |  |  |

На [рис. 4.20](https://www.intuit.ru/studies/courses/494/350/lecture/8358?page=7#image.4.20). вы можете видеть форму frm\_MovingButton.



**Рис. 4.20.**Форма задачи об убегающей кнопке в окне редактора VBA

Дважды щелкнем по cmd\_Yes. В редакторе кода выберем в поле событий MouseMove.

В тело обработчика события MouseMove для кнопки cmd\_Yes введите такой текст ([листинг 4.16](https://www.intuit.ru/studies/courses/494/350/lecture/8358?page=7#example.4.16)).

cmd\_Yes.Left = Rnd \* 250

cmd\_Yes.Top = Rnd \* 150

Листинг 4.16. Код обработчика MouseMove кнопки cmd\_Yes

В листинге 4.16. мы обращаемся к следующим свойствам кнопки cmd\_Yes:

* Left — расстояние между кнопкой и левым краем формы.
* Top — расстояние между кнопкой и верхним краем формы.

Функция *Rnd* возвращает случайное число от 0 до 1. Для вычисления нового расстояния от кнопки до левого края формы умножаем случайное число на 250, до верхнего края формы – на 150. Эти значения взяты не случайно. Ниже приведен их расчет.

250 = (ширина формы) – (ширина кнопки) – (запас по ширине)= 350-80-20

150 = (высота формы) – (высота кнопки) – (запас по высоте) = 200 – 30 – 20

Каждый раз, когда пользователь наводит указатель мыши на кнопку cmd\_Yes, кнопка случайным образом меняет положение на форме. Значит, с помощью мыши он не сможет нажать на кнопку.

Кнопки на формах можно нажимать не только с помощью мыши. Если на кнопке установлен фокус ввода (как вы знаете, он перемещается по элементам управления по нажатию клавиши Tab на клавиатуре), "нажать" на кнопку можно, нажав клавишу Пробел или Enter. Свойство Tabstop кнопки cmd\_Yes мы установили равным False — то есть кнопка не сможет получить фокус ввода и ее нельзя будет нажать даже с использованием клавиатуры.

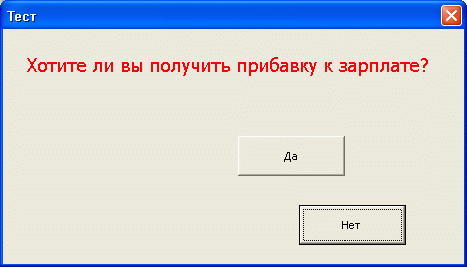
Для того чтобы программа приобрела законченный вид, добавим обработчики события Click для кнопок cmd\_No и cmd\_Yes. Нажатие на кнопку cmd\_No должно вывести надпись: "Спасибо за участие в опросе" и завершить программу. А если пользователю все же удастся нажать на неуловимую cmd\_Yes (это возможно, если кнопка "прыгнет" так, что окажется под указателем мыши), должна вывестись какая-нибудь подходящая надпись.

Для удобства работы с программой добавьте на лист MS Excel кнопку cmd\_Show\_Form, которая вызывала бы форму программы. Используйте метод Show формы frm\_MovingButton, чтобы отобразить ее по нажатию на кнопку ([листинг 4.17](https://www.intuit.ru/studies/courses/494/350/lecture/8358?page=7#example.4.17)).

Frm\_MovingButton.Show

Листинг 4.17. Код обработчика Click кнопки cmd\_Show\_Form

На [рис. 4.21](https://www.intuit.ru/studies/courses/494/350/lecture/8358?page=7#image.4.21). вы можете видеть запущенную форму программы в процессе работы с ней на фоне листа Microsoft Excel с кнопкой для вызова этой формы.



**Рис. 4.21.**Программа с убегающей кнопкой в действии

**4.22. Выводы**

В этой лекции мы познакомились с особенностями конструирования пользовательских интерфейсов при помощи элементов управления. Однако создать *интерфейс* – это лишь часть работы программиста. Другая часть заключается в написании кода. Основами кодирования на *VBA* мы займемся в следующей лекции.

Лекция 5. Основы VBA: синтаксис, переменные, типы данных

**5.1. Синтаксические описания**

*Синтаксис* языка программирования — это правила написания программ.

Рассмотрим пример — возьмем из справочной системы описание команды. Мы пока не будем вдаваться в смысл ключевых слов (перед нами — описание *оператора* *цикла*), приведенных здесь, остановимся на основных частях описания.

For counter = start To end [Step step]

[statements]

[Exit For]

[statements]

Next [counter]

В синтаксических описаниях *VBA* жирным шрифтом выделены ключевые слова языка, курсивом даны названия *переменных* и других *элементов языка*. В квадратных скобках располагаются необязательные элементы – то есть такие элементы, без описания которых можно обойтись. Например, в приведенном описании блоки [statements] будут содержать в себе "полезную нагрузку" *цикла* – *операторы*, которые будут выполняться многократно, однако цикл будет работать и без них.

Элементы, которые допускают альтернативный выбор, разделяются знаком " | ". Например, такая *запись*: " one | two " обозначает "*one* или two".

Ниже мы будем останавливаться на существенно важных особенностях тех или иных конструкций. Если вам понадобятся подробности — вы сможете найти их в справке к *VBA*.

Теперь рассмотрим понятие *переменной* — важнейшее понятие в любом языке программирования.

**5.2. Переменные**

[05-01-Переменная.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/5/files/05_01.docm) — пример к п. 5.2.

***Переменная — это именованная область памяти, где могут храниться различные данные, которые можно изменять во время выполнения программы***. *Переменные* — это одна из основ любой программы. В них можно сохранять введенную пользователем информацию, их можно использовать для накопления данных, обработанных в программе и так далее. Если бы не было *переменных* — программирования не существовало бы.

У *переменной* есть несколько важнейших характеристик. Первая — имя *переменной*. Используя имя, мы можем обращаться к *переменной* в программе. Вторая характеристика — это *тип данных*, которые могут храниться в *переменной*. Тип определяет характер данных, которые мы можем хранить в *переменной*. Например, это могут быть числовые данные (возраст пользователя) и строковые данные (*имя пользователя*).

О типах мы поговорим ниже, а пока займемся именами *переменных*. Давая *переменным* имена, следует придерживаться следующих правил.

* Имя *переменной* должно состоять из букв и цифр, причем оно должно начинаться с буквы.
* Имя не может быть длиннее 255 символов
* Имя не должно содержать специальных знаков ( #, $, % и т.д.) и пробелов.
* В качестве имен нельзя использовать зарезервированные слова VBA (например – if, dim, for и т.д.).

При именовании *переменных* старайтесь пользоваться латинскими буквами. Иначе возможны проблемы при работе ваших программ на различных версиях Office.

Например, такие имена *переменных* выглядят вполне корректно: str\_Name, num\_Age, str\_Name1, num\_Item2 и т.д. Обратите внимание на префиксы, которыми мы снабдили имена *переменных*. Это признак особого стиля именования *переменных*. Он очень напоминает стиль именования элементов управления, который мы рассматривали выше.

Напишем программу на *VBA*, иллюстрирующую простой пример работы с *переменными*. Сначала она должна задать пользователю вопрос: "Как вас зовут", а потом — вывести его имя в окне сообщения.

Для этого добавим в документ Microsoft *Word* кнопку с именем cmd\_Variables и надписью "*Имя пользователя*". Зададим ее обработчику Click следующий код ([листинг 5.1](https://www.intuit.ru/studies/courses/494/350/lecture/8360?page=1#example.5.1).):

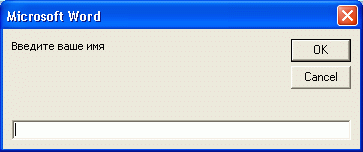
txt\_Name = InputBox("Введите ваше имя")

MsgBox (txt\_Name)

Листинг 5.1. Код обработчика Click кнопки cmd\_Variables

Первая строка этого кода содержит новую для вас функцию InputBox. Она используется для вывода окна с сообщением и полем для ввода данных. Строка, которая передается функции в качестве параметра — это и есть сообщение, которое будет выведено в окне.

На [рис. 5.1](https://www.intuit.ru/studies/courses/494/350/lecture/8360?page=1#image.5.1). вы можете видеть это окно.



**Рис. 5.1.**Окно ввода данных

*Выражение* txt\_Name = означает, что мы присваиваем *переменной* с именем txt\_Name то, что *пользователь* введет в окне ввода данных, как бы говорим программе: "Сохрани в *переменной* txt\_Name то, что *пользователь* введет в окне". Когда *пользователь* сообщает свое имя и нажимает **OK** —введенная им строка записывается в *переменную* txt\_Name. Введем на приглашение программы имя "Александр", вы можете ввести любое другое имя.

Теперь мы можем выполнять много разных действий с этой *переменной*. В частности — использовать в качестве параметра, передаваемого функции MsgBox. В данном примере мы так и поступили — вызвали MsgBox, указав вместо выводимого текста имя *переменной*. В результате в окне сообщения вывелось *имя пользователя*, введенное на предыдущем этапе ([рис. 5.2](https://www.intuit.ru/studies/courses/494/350/lecture/8360?page=1#image.5.2).).



**Рис. 5.2.**Имя пользователя в окне сообщения

Обратите внимание на то, что мы не заключили имя *переменной* в кавычки. Иначе в окне сообщения вывелось бы лишь имя *переменной* ( txt\_Name ), но не хранимый в ней текст.

**5.3. Типы данных в VBA**

*Тип данных* определяет важнейшие свойства *переменной*. А именно, следующее:

* что может храниться в *переменной* (текст, число, некоторые другие виды данных);
* размер памяти, необходимый для хранения *переменной* (измеряется в байтах);
* операции, которые можно производить с *переменной* (например, невозможно извлечь квадратный корень из слова "Привет" так как эта операция не определена для строковых данных);

Вы уже можете предположить как *минимум* два *типа данных*, которые могут хранить *переменные* — строковые и числовые. На самом деле *список* типов данных *VBA* гораздо обширнее.

В [табл. 5.1](https://www.intuit.ru/studies/courses/494/350/lecture/8360?page=1#table.5.1). вы найдете информацию об основных типах данных *VBA*.

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица 5.1. Типы данных в VBA | | | |
| **Тип данных** | **Размер, байт** | **Описание** | **Диапазон значений** |
| Variant | 16 (числа) 22+длина строки (строки) | Может хранить данные любых типов |  |
| Integer | 2 | Целое число | от -32768 до 32767 |
| Long | 4 | Длинное целое | от -2147483648 до 2147483647 |
| Single | 4 | *Число с плавающей запятой* | Для отрицательных: от -3.402823E38 до -1.401298E-45 Для положительных: от 1.401298E-45 до 3.402823E38 |
| Double | 8 | *Число с плавающей запятой* двойной точности, для отрицательных значений и для положительных | Для отрицательных от -1.79769313486231E308 до -4.94065645841247E-324 Для положительных: от 4.94065645841247E-324 до 1.79769313486232E308 |
| Byte | 1 | Байт | от 0 до 255 |
| *Currency* | 8 | Денежный формат | от -922,337,203,685,477.5808 до 922,337,203,685,477.5807 |
| Decimal | 14 | Масштабируемое целое | 29-значное число с 28 знаками справа от запятой |
| Boolean | 2 | Логический | True или False |
| Date | 8 | Дата и время | от 1 января 100 г. до 31 декабря 9999 г. |
| Object | 4 | Ссылка на объект | Различные виды объектов |
| String | Зависит от длины строки | Строка |  |

Как видите, в *VBA* существует немало типов данных. Наверное, сейчас вам кажется, что их слишком много. Может показаться, что нет ничего удобнее *типа данных* Variant. Однако надо учесть, что за *универсальность* типа Variant приходится платить производительностью и системными ресурсами. Чем больше места требуется для хранения *переменной* — тем ниже скорость работы с ней.

Если вы хотите, чтобы ваши программы работали как можно быстрее — выбирайте *типы данных*, которые используют для хранения вашей информации *минимум* системных ресурсов. Если вы сомневаетесь, хватит ли размерности выбранного *типа данных* для решения ваших задач — возьмите более емкий тип.

Как правило, для работы с дробными числами и различных вычислений используют *тип данных* Double. Для *переменных*, которые используются в циклах, счетчиках — Integer или Long. В работе со строками используют String. Объектные *переменные* имеют самые разные типы, которые определяются объектами, для обращения к которым они создаются. В небольших проектах вполне оправдано использование *переменных* типа Variant – это не слишком замедлит работу. Однако, если размер проекта растет или вы нуждаетесь в ускорении ресурсоемких вычислений – подходите к выбору типов данных ответственно.

Если при объявлении *переменной* не указывать ее тип — он автоматически устанавливается в Variant. Объявляя *переменную* без указания типа, вы пользуетесь преимуществами работы с объявленными *переменными*, и, в то же время, можете гибко использовать *переменную* для хранения различных типов данных (например – чисел с плавающей запятой и целых чисел). Такой подход — объявление *переменных* без указания типа — допустим при создании небольших проектов, а так же — на начальном этапе работы над достаточно масштабными программами, когда вы не знаете точно, какой *тип данных* понадобится для той или иной *переменной*. В листингах, которые вы найдете ниже, мы явно задавали тип *переменных* лишь тогда, когда это либо требовалось *по* ходу выполнения примера (например – типы объектных *переменных* ), либо, когда это нужно для лучшего понимания материала (*числовой тип* для возраста, строковой — для имени и т.д.) В основном же мы использовали *переменные* типа Variant.

Выше мы уже говорили о венгерской нотации как о способе именования объектов. Те же правила действуют и для *переменных*. *Префикс* имени обычно говорит о типе *переменной*. В [табл. 5.2](https://www.intuit.ru/studies/courses/494/350/lecture/8360?page=1#table.5.2). приведены некоторые префиксы.

|  |  |
| --- | --- |
| Таблица 5.2. Префиксы и типы переменных | |
| **Префикс имени переменной** | **Пояснения** |
| s, str | Строковая переменная |
| n, num | Числовая переменная (существует множество числовых типов данных, название каждого из которых может быть использовано в качестве префикса, например, int для Integer, dbl для Double и т.д.) |
| b, bool | *Логическая переменная* |
| O, obj | *Объектная переменная* |

Для работы с *переменными* можно использовать два подхода. При первом мы выбираем имя *переменной* и используем ее в программе без каких-либо подготовительных действий. Именно так мы работали с *переменной* в примере к п. 5.2. Такая *переменная* приобретет тип Variant. Второй подход предусматривает объявление *переменных* перед использованием.

**5.4. Преимущества объявления переменных**

Как вы уже знаете, в *VBA* *переменные* можно использовать без предварительного объявления и указания типа. Но такой подход допустим лишь при разработке небольших программ, состоящих буквально из пары десятков строк. Когда *программа* увеличивается, контролировать работу с *переменными*, которые спонтанно появляются в коде, становится сложно.

Необъявленные *переменные* могут вызывать трудно диагностируемые ошибки. Например, рассмотрим такой код ([листинг 5.2](https://www.intuit.ru/studies/courses/494/350/lecture/8360?page=2#example.5.2).):

txt\_CompanyName = "Типография"

MsgBox (txt\_CompanyName)

Листинг 5.2. Пример работы с переменной

Как вы думаете, что получится при его исполнении? Очевидно, должно вывестись окно сообщения с текстом "Типография". Но это далеко не факт. *По* внешнему виду *переменных* можно предположить, что их имена написаны латинскими буквами. Предположим, что в строке, где мы присваиваем *переменной* *значение* "Типография", это так и есть. А во второй строке мы ошиблись — и вместо латинской написали русскую букву "С" (они и на клавиатуре занимают одну клавишу, а внешне различить их просто невозможно). Как результат – совершенно непонятная ошибка – все выглядит правильно, а работать не хочет.

Если бы *редактор кода* не давал нам пользоваться необъявленными *переменными* — подобного рода ошибки были бы пресечены на корню. Чтобы заставить редактор автоматически требовать объявление *переменных*, можно поступить одним из двух способов. Первый — вставить в раздел объявлений модуля (то есть — вне кода процедур и обработчиков событий) команду Option *Explicit*. Второй — включить запрещение работы с необъявленными *переменными* в настройках редактора. Для этого выполните команду главного *меню* Tools • Options (Инструменты • Опции) и в появившемся диалоговом окне на вкладке Editor (Редактор) включите *параметр* Require *Variable Declaration* (Требовать объявление *переменных* ).

**5.5. Объявляем переменные: Dim и Static**

[05-02-Dim и Static.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/5/files/05_02.docm) — пример к п. 5.5.

Лучше всего объявлять *переменные* перед использованием. Так вы избежите ненужных ошибок, и, при работе с объектными *переменными*, сможете получать подсказки при их использовании.

Рассмотрим пример. Объявление *переменной* с именем num\_MyAge типа Byte выглядит так ([листинг 5.3](https://www.intuit.ru/studies/courses/494/350/lecture/8360?page=2#example.5.3).):

Dim num\_MyAge As Byte

num\_MyAge = 23

Листинг 5.3. Объявление переменной и работа с ней

Этот код можно перевести так: "Объявить *переменную* типа Byte с именем num\_MyAge и сделать ее доступной в модуле, в котором она объявлена". Второй строкой мы присваиваем *переменной* число 23.

Доступность *переменной*, или, как говорят, *область видимости*, задается ключевым словом Dim. Оно означает, что мы сможем пользоваться объявленной *переменной* только внутри того модуля, в котором она объявлена.

Обратите внимание на то, что если вы присваиваете *значение* строковой *переменной* — передаваемое ей *значение* должно быть заключено в кавычки. При присваивании значений даты соответствующим *переменным*, эти значения должны быть заключены в значки #. Например, так ([листинг 5.4](https://www.intuit.ru/studies/courses/494/350/lecture/8360?page=2#example.5.4).):

Dim str\_MyName as String

Dim date\_MyBirthDate as Date

str\_MyName = "Alexander"

date\_MyBirthDate = #January 1, 2007#

Листинг 5.4. Работа с переменными типа String и Date

Помимо ключевого слова Dim могут использоваться и другие слова. В частности, нам будет полезна *команда* Static — *переменная*, объявленная с использованием этой команды, будет хранить свое *значение* между вызовами процедуры, область ее видимости аналогична *переменной*, объявленной с Dim. Такую *переменную* можно использовать для накопления каких-либо значений.

Рассмотрим пример. Создайте новый документ Microsoft *Word*, добавьте в него две кнопки. В обработчике события Click одной из них с именем cmd\_Dim, на которой будет написано Dim, напишите такой код ([листинг 5.5](https://www.intuit.ru/studies/courses/494/350/lecture/8360?page=2#example.5.5).)

Dim num\_Counter As Integer

num\_Counter = num\_Counter + 1

MsgBox (num\_Counter)

Листинг 5.5. Обработчик события Click кнопки cmd\_Dim

Здесь мы сначала объявляем *переменную* типа Integer, потом присваиваем этой *переменной* ее же *значение*, увеличенное на 1, а дальше — выводим *переменную* в окне сообщения.

Сколько бы вы не нажимали на эту кнопку — всякий раз в окне сообщения будет выводиться *единица*. Ведь каждый раз, когда запускается обработчик, все происходит как бы в первый раз — *информация* об изменениях *переменной* не сохраняется при выходе из обработчика.

Вторую кнопку назовите cmd\_Static, подпишите ее как Static, и в обработчик нажатия добавьте следующее ([листинг 5.6](https://www.intuit.ru/studies/courses/494/350/lecture/8360?page=2#example.5.6).):

Static num\_Counter As Integer

num\_Counter = num\_Counter + 1

MsgBox (num\_Counter)

Листинг 5.6. Обработчик события Click кнопки cmd\_Static

Единственное отличие этого текста от предыдущего — *ключевое слово* Static вместо Dim. Но благодаря такому объявлению, *переменная* хранит свое *значение* между запусками обработчика. Попробуйте несколько раз нажимать на кнопку Static — каждый раз вы будете видеть в окне сообщения число, увеличенное на 1.

**5.6. Арифметические операторы и работа с числовыми переменными**

[05-03-Операторы, конкатенация.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/5/files/05_03.docm) — пример к пп. 5.6.-5.7.

*Операторы* — это команды, которые используются в *VBA* для работы с данными. Если вы программировали когда-нибудь, например, на Basic'e или на каком-нибудь другом языке, вам уже знакомы основные *операторы*. Ну а если вы раньше не сталкивались с *операторами* в программировании, то уж со школьным курсом математики вы сталкивались точно. Большинство *операторов* *VBA* действуют точно так же, как знакомые всем арифметические *операторы*. В [табл. 5.3](https://www.intuit.ru/studies/courses/494/350/lecture/8360?page=2#table.5.3). вы можете видеть описание арифметических *операторов*, применимых в *VBA*

|  |  |
| --- | --- |
| Таблица 5.3. *Арифметические операторы* VBA | |
| **Оператор** | **Описание** |
| + | Сложение. Например, в результате вычисления выражения 4+3 получится 7. |
| – | Вычитание. |
| \* | Умножение |
| / | Деление |
| ^ | Возведение в степень, 5^2 равняется 25 |
| \ | Целочисленное деление. От результата деления отбрасывается дробная часть. Например, 10 \ 3 равняется 3. |
| mod | Деление по модулю. Возвращает остаток от деления. Например, 10 mod 3 равняется 1 |

В *VBA* применяется тот же порядок обработки выражений, который принят в математике. Например, результат вычисления 2\*3+7 равняется 13-ти. Так же, в математических выражениях могут использоваться круглые скобки. Однако, круглые скобки могут встретиться вам не только в математических выражениях. Скобки, как вы могли заметить из примеров, которые мы рассматривали выше, часто употребляются при вызове функций *VBA*, при работе с объектами и т.д.

Если снова вернуться к математике, нельзя не заметить, что редкая математическая *запись* обходится без знака =. Этот знак используется в *VBA* в нескольких ролях.

Во-первых, это *оператор присваивания*. Вы уже сталкивались с ним, когда присваивали какие-то значения свойствам элементов управления или *переменным*. Например, такая *запись*: num\_a = 10 + 2 означает: "Присвоить *переменной* num\_a результат сложения чисел 10 и 2 " или, проще, "а равно 10+2". Вторая роль *оператора* = заключается в том, что он используется в командах сравнения выражений — о них мы поговорим в следующей главе.

**5.7. Конкатенация**

Несколько ролей и у *оператора* +. Во-первых — это *арифметический оператор* сложения. А во-вторых — оператор конкатенации строк. *Конкатенация* — это нечто вроде "склеивания" строк. В качестве *оператора* конкатенации можно использовать и оператор &. Считается, что & использовать предпочтительнее так как он в любом случае обрабатывает операнды как строковые данные.

Давайте рассмотрим пример, который охватывает арифметические *операции* и конкатенацию строк.

Напишем программу, которая запрашивает *имя пользователя* и два числа, после чего выводит такой текст в окне сообщения (если имя введено как "Александр", первое число 2, второе – 21): "Здравствуйте, Александр. Вы ввели числа 2 и 21, их сумма равняется 23."

Добавим кнопку в документ Microsoft *Word*, назовем ее cmd\_Experiments, надпишем ее как Работа с *операторами*.

Теперь решим, какие *переменные* нам нужны.

Для имени пользователя это *переменная* типа String. Дадим ей имя str\_UserName. Для чисел нам понадобится пара *переменных* одного из числовых типов.

Какие числа введет *пользователь*? Этого мы не знаем. Конечно, можно ограничить ввод проверками, но это дела будущих примеров. Предполагается, что *пользователь* может ввести практически любое число — целое или дробное, положительное или отрицательное. Поэтому воспользуемся типом Double. Назовем пару числовых *переменных* num\_First и num\_Second.

Создадим отдельную *переменную* того же типа Double для хранения суммы введенных значений – назовем ее num\_Summ.

В условии нашей задачи присутствует требование вывести все введенные данные в строку, скомбинировав с определенными словами. Нам понадобится *переменная* типа String для хранения этой строки. Назовем ее str\_Result.

Также добавим в обработчик нажатия кнопки команду Option *Explicit* — тогда система запретит использование необъявленных *переменных*.

Вы уже знакомы с функцией InputBox, которая выводит окно для ввода данных пользователем. Как вы могли убедиться, InputBox отлично справляется со вводом строковых значений. А как насчет чисел? Можем ли мы написать что-то вроде:

num\_First = InputBox("Введите первое число")

Вполне можем.

Пожалуй, сейчас для вас самое сложное — это построить строку вывода, однако пользуйтесь следующими правилами и вы всегда будете правильно и быстро строить подобные строки.

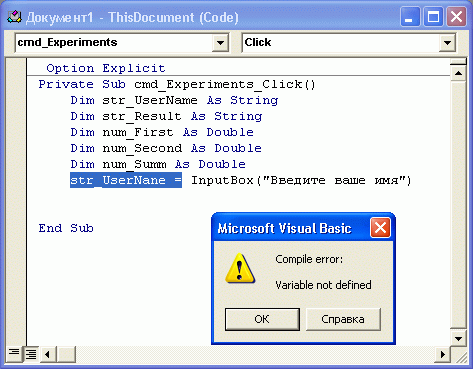
Во-первых — помните, что все, что вы хотите вывести в виде неизменного текста, должно быть включено в кавычки, а имена *переменных*, наоборот, пишутся без кавычек. Во-вторых — используйте оператор & при "склеивании" отдельных частей строки или оператор + и функцию Str (о ней вы прочтете ниже), которая конвертирует числовые *переменные* в строки. Так вы будете гарантированы от неожиданностей и ошибок.

Например, пусть в *переменной* str\_UserName хранится *имя пользователя* "Александр", а *переменная* str\_Result должна содержать результат вывода. Напишем код, помещающий в str\_Result строку "Привет, Александр":

str\_Result = "Привет, " & str\_UserName

Видите? Ничего сложного, правда?

Обратите внимание на то, как реагирует система на использование необъявленных *переменных* при добавленной в *модуль* команде Option *Explicit* ([рис. 5.3](https://www.intuit.ru/studies/courses/494/350/lecture/8360?page=2#image.5.3).).



**Рис. 5.3.**Реакция системы на необъявленную переменную

Здесь вместо str\_UserName мы ошибочно использовали str\_UserNane. Но благодаря Option *Explicit* появление необъявленных *переменных* воспринимается как ошибка. При попытке запуска программы мы видим *сообщение об ошибке*: "Variable *not* defined" - "*Переменная* не определена".

Однако вернемся к нашей задаче. Мы определились с особенностями работы наиболее сложных ее частей, в итоге у нас получилось следующее ([листинг 5.7](https://www.intuit.ru/studies/courses/494/350/lecture/8360?page=2#example.5.7).):

Dim str\_UserName As String

Dim str\_Result As String

Dim num\_First As Double

Dim num\_Second As Double

Dim num\_Summ As Double

str\_UserName = InputBox("Введите ваше имя")

num\_First = InputBox("Введите первое число")

num\_Second = InputBox("Введите второе число")

num\_Summ = num\_First + num\_Second

str\_Result = "Здравствуйте, " & str\_UserName \_

& ". Вы ввели числа " & num\_First & " и " & num\_Second \_

& ". Их сумма равняется " & num\_Summ

MsgBox (str\_Result)

Листинг 5.7. Обработчик события Click кнопки cmd\_Experiments

Обратите внимание на то, что строку сборки строки вывода str\_Result мы разбили на несколько частей *оператором* переноса строки.

**5.8. Преобразование типов данных**

[05-04-Эксперимент с типами данных.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/5/files/05_04.docm)— пример к п. 5.8.

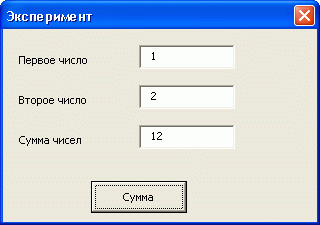
Нередко возникают ситуации, когда данные одного типа надо преобразовать в данные другого. Ярче всего это проявляется тогда, когда нужно преобразовать число, представленное в виде строки и хранящееся в строковой *переменной*.

Давайте проведем эксперимент. Создадим форму, назовем ее frm\_First, разместим на ней три текстовых поля — txt\_First и txt\_Second для ввода чисел, и txt\_Summ для вывода суммы этих чисел. Добавим на форму кнопку cmd\_First с надписью Сумма и создадим для нее обработчик события Click ([листинг 5.8](https://www.intuit.ru/studies/courses/494/350/lecture/8360?page=3#example.5.8).).

txt\_Summ = txt\_First + txt\_Second

Листинг 5.8. Обработчик события Click кнопки cmd\_First

На [рис. 5.4](https://www.intuit.ru/studies/courses/494/350/lecture/8360?page=3#image.5.4). вы можете видеть форму после того, как мы ввели в первое и второе поля числа 1 и 2 и нажали на кнопку.



**Рис. 5.4.**Эксперимент: вместо сложения — конкатенация

Как видите, вместо сложения мы получили конкатенацию. Любые значения, вводимые в текстовые поля, *по* умолчанию рассматриваются как строковые. Чтобы все-таки получить сумму введенных чисел в *поле* "Сумма чисел", нам нужно превратить строки, пусть и содержащие числовые символы, в настоящие числа, преобразовать их из *типа данных* String в один из числовых типов.

Для таких "превращений" существуют специальные функции *преобразования типов*.

**5.9. Val — тип String в тип Double**

*Функция* Val применяется для конверсии строковых *переменных* в числовые, а именно – *переменных* типа String в тип Double.

Чтобы наш пример заработал, код обработчика нажатия кнопки можно переписать так ([листинг 5.9](https://www.intuit.ru/studies/courses/494/350/lecture/8360?page=3#example.5.9).):

txt\_Summ = Val(txt\_First) + Val(txt\_Second)

Листинг 5.9. Измененный обработчик события Click кнопки cmd\_First

Давайте рассмотрим еще несколько примеров использования этой функции.

Val (" 12345привет") возвратит число 12345.

Val читает предлагаемую ей строку слева направо, игнорируя пробелы. Она считывает все числовые знаки до первого символьного знака и преобразует считанное в число. В качестве дробных символов *функция* понимает лишь точки.

Val ("1 2 3") возвратит число 123

Val ("1 2 и 3") возвратит число 12.

Иногда нужно провести *обратное преобразование* — превратить число в строку.

**5.10. Str — числовые типы в String**

*Функция* Str конвертирует данные различных числовых типов в тип String.

Особенность функции заключается в том, что первый символ полученного строкового значения зарезервирован для знака числа. Если в строку конвертируется число отрицательное — первый символ полученной строки — знак -. Если конвертируется положительное число, первым символом полученной строки будет *пробел*, а дальше будут идти числовые символы.

Например, *функция* Str (12) возвратит строку " 12 ". Мы рассмотрим пример с использованием функции Str немного ниже, когда будем говорить о работе со строками.

Существуют и другие функции, предназначенные для конверсии типов данных. Их названия состоят из сокращенного слова "*Convert*" и сокращенного же названия *типа данных*, в который они конвертируют входные значения. Например, это CBool, CByte, CCur, CDate, CDbl, CDec, CInt, CLng, CSng, CStr, CVar. Скажем, *функция* CInt конвертирует данные в формат Integer. Учитывая особенности этого *типа данных*, корректно могут быть сконвертированы лишь значения от -32768 до 32767. Причем, дробные числа округляются при конверсии до ближайшего четного числа — 0.5 округляется до 0, 1.5 — до 2. Если вам понадобятся подробности о каждой из этих функций — обратитесь к справочной системе *VBA*.

**5.11. Функции проверки типа данных**

[05-05-Проверка типов данных.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/5/files/05_05.docm)— пример к п. 5.11.

Если вам нужно узнать *тип данных* *переменной*, вы можете воспользоваться функцией TypeName.

Добавим в документ Microsoft *Word* кнопку, назовем ее cmd\_DataType, надпишем как Проверка типа и внесем в ее обработчик Click такой код ([листинг 5.10](https://www.intuit.ru/studies/courses/494/350/lecture/8360?page=3#example.5.10).):

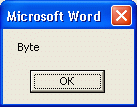
Dim num\_MyAge as Byte

num\_MyAge = 24

MsgBox (TypeName(num\_MyAge))

Листинг 5.10. Обработчик события Click кнопки cmd\_DataType

На ([рис. 5.5](https://www.intuit.ru/studies/courses/494/350/lecture/8360?page=3#image.5.5).) вы можете видеть результат выполнения этого кода.



**Рис. 5.5.**Результат вывода типа данных

Чтобы проверить, являются ли данные, хранимые в *переменной* типа Variant, числом, можно воспользоваться функцией IsNumeric.

Для точного определения *типа данных*, которые хранятся в *переменной* типа Variant, вы можете воспользоваться функцией VarType.

**5.12. Выводы**

В этой лекции мы подробно обсудили особенности использования различных типов данных в *VBA*, работу с *переменными*, *операторами* и функциями *преобразования* и проверку типов данных. Наша следующая лекция посвящена *встроенным* и *пользовательским функциям*.

Лекция 6. Основы VBA: встроенные и пользовательские функции

**6.1. Встроенные математические функции**

[06-01-Математика.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/6/files/06_01.docm) - пример к п. 6.1.

Вы хотите вычислить квадратный корень, округлить число или сделать с ним еще что-нибудь подобное? Для этого *VBA* имеет специализированные функции, вы можете найти их в [табл. 6.1](https://www.intuit.ru/studies/courses/494/350/lecture/8361?page=1#table.6.1).

|  |  |
| --- | --- |
| Таблица 6.1. Встроенные математические функции | |
| **Функция** | **Описание** |
| Abs | Абсолютное значение |
| Atn | Арктангенс |
| Cos | Косинус числа |
| Exp | Возвращает число e (2.718282), возведенное в степень аргумента функции. |
| Fix | Отбрасывает дробную часть числа и возвращает целую. В результате для положительных чисел получается число меньшее, чем входное (Fix(2.5) возвратит 2), для отрицательных - большее (Fix(-2.5) возвратит -2) |
| Int | Отбрасывает дробную часть числа и возвращает целую. Для положительных получается число меньшее введенного (Int(2.5) возвратит 2), для отрицательных - так же меньшее (Int(-2.5) возвратит -3). |
| Log | Возвращает *натуральный логарифм* числа |
| *Rnd* | Возвращает случайное число типа Single, причем, это число находится между 0 и 1. Для инициализации генератора случайных чисел используйте директиву Randomize - ее надо вызвать до вызова *Rnd*. |
| Sgn | Функция предназначена для определения знака числа. Если число положительное - она возвращает 1. Для нуля функция возвратит 0, для отрицательного числа -1. |
| Sin | Синус |
| Sqr | Квадратный корень |
| Tan | Тангенс |

Давайте рассмотрим пример. Добавим в документ Microsoft *Word* кнопку, назовем ее cmd\_Calc, надпишем ее как Вычисления и добавим следующий код ([листинг 6.1](https://www.intuit.ru/studies/courses/494/350/lecture/8361?page=1#example.6.1).), иллюстрирующий работу рассмотренных функций.

Dim dblNumber As Double

'Переменная, используемая в вычислениях

Dim varResult

'Переменная типа Variant

dblNumber = Val(InputBox("Введите число"))

'Вычисляем абсолютное значение введенного числа

'Сначала присвоим результат переменной varResult

'Далее - выведем подписанный результат в окне

'сообщения, воспользуемся знаком "+" для

'конкатенации строк, в других случаях

'будем вызывать функции непосредственно

'в MsgBox'e

'Обратите внимание на то, что мы конвертируем

'числовые значения в строки с помощью функции Str

varResult = Abs(dblNumber)

MsgBox ("Абсолютное значение " + \_

Str(dblNumber) + " равняется " + Str(varResult))

'Арктангенс

MsgBox ("Арктангенс " + \_

Str(dblNumber) + " равняется " + \_

Str(Atn(dblNumber)))

'Косинус

MsgBox ("Косинус " + \_

Str(dblNumber) + " равняется " + \_

Str(Cos(dblNumber)))

'e в степени введенного числа

MsgBox ("Число e в степени " + \_

Str(dblNumber) + " равняется " + \_

Str(Exp(dblNumber)))

'Функция Fix

MsgBox ("Результат работы функции Fiх для " + \_

Str(dblNumber) + " равняется " + \_

Str(Fix(dblNumber)))

'Функция Int

MsgBox ("Результат работы функции Int для " + \_

Str(dblNumber) + " равняется " + \_

Str(Int(dblNumber)))

'Натуральный логарифм

MsgBox ("Натуральный логарифм " + \_

Str(dblNumber) + " равняется " + \_

Str(Log(dblNumber)))

'Получим несколько случайных чисел

'первое число - от 0 до 1

'второе - от 0 до 10.

'Третье - от 25 до 100

'Четвертое - целое от 0 до 34

Randomize

MsgBox ("Группа случайных чисел: " + \_

Str(Rnd()) + ", " + \_

Str(Rnd() \* 10) + ", " + \_

Str(Rnd() \* 75 + 25) + ", " + \_

Str(Int(Rnd() \* 34)))

'Функция Sgn

MsgBox ("Результат работы Sgn для " + \_

Str(dblNumber) + " равняется " + \_

Str(Sgn(dblNumber)))

'Cинус

MsgBox ("Синус " + \_

Str(dblNumber) + " равняется " + \_

Str(Sin(dblNumber)))

'Квадратный корень

MsgBox ("Квадратный корень " + \_

Str(dblNumber) + " равняется " + \_

Str(Sqr(dblNumber)))

'Тангенс

MsgBox ("Тангенс " + \_

Str(dblNumber) + " равняется " + \_

Str(Tan(dblNumber)))

Листинг 6.1. Обработчик события Click кнопки cmd\_Calc

Обратите внимание на *алгоритм* получения случайного числа, находящегося в определенном диапазоне, с помощью функции *Rnd*. Предположим, нам нужно получить случайное число от 15 до 40. Получим, для начала, число от 0 до 40. Очевидно, что для этого нам понадобится такой вызов: *Rnd*()\*40.

Чтобы "поднять" уровень наименьшего случайного числа, возвращаемого выражением, до 15, сделаем следующее.

Во-первых, вычислим *разность* 40 и 15 - у нас получится 25. Значит, чтобы получить случайное число от 0 до 25, можно использовать вызов *Rnd*()\*25.

Во-вторых, прибавим к полученному случайному числу 15. Теперь *выражение* для получения случайного числа от 15 до 40 выглядит так: *Rnd*()\*25+15.

Проверим это *высказывание* на правильность. *Функция* *Rnd*, как известно, возвращает случайные числа от 0 до 1. Если *функция* возвратит 0 - результат вычисления выражения будет равен 15 (0\*25+15). Если *функция* возвратит 1 - результат будет равен 40 (25\*1+15). Промежуточные значения *Rnd* дадут искомые случайные числа между 15 и 40.

На [рис. 6.1](https://www.intuit.ru/studies/courses/494/350/lecture/8361?page=1#image.6.1). вы можете видеть окно сообщения, содержащее результаты вызовов функции *Rnd*.



**Рис. 6.1.**Окно сообщения, выводящее случайные числа

Выше мы работали, в основном, с числами, теперь поговорим о строковых переменных.

**6.2. Строковые функции**

[06-02-Строковые функции.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/6/files/06_02.docm)- пример к п. 6.2.

В [табл. 6.2](https://www.intuit.ru/studies/courses/494/350/lecture/8361?page=1#table.6.2). вы можете найти информацию об основных *строковых функциях* *VBA*.

|  |  |
| --- | --- |
| Таблица 6.2. *Строковые функции* | |
| **Функция** | **Описание** |
| Len(string) | Возвращает длину строки. Например, длина строки "Добрый день" составляет 11 символов - учитывая пробел. Выходное значение имеет тип Long |
| LCase(string) | Возвращает строку, все символы которой записаны в нижнем регистре. Например, строка "Привет" превратится в "привет" |
| UCase(string) | Возвращает строку, все символы которой записаны в верхнем регистре. Например, для "Привет" мы получим "ПРИВЕТ" |
| String(number, character) | Возвращает строку, состоящую из number символов character |
| Left(*string, length*) | Возвращает length символов, начиная с первого левого символа строки string |
| Right(*string, length*) | Возвращает length символов, начиная с самого правого символа строки string |
| LTrim(string) | Возвращает строку, в которой вырезаны все пробелы слева |
| RTrim(string) | Вырезает из строки все пробелы справа |
| Trim(string) | Вырезает из строки все пробелы слева и справа |
| Mid(string, start[, length]) | Вырезает из строки string с позиции start length символов |
| *Asc*(string) | Возвращает ASCII-код первого символа строки |
| *Chr*(charcode) | Возвращает символ, соответствующий коду символа |

Работа со строками традиционно считается сложным разделом программирования для начинающих, поэтому проиллюстрируем вышеописанные функции примерами.

Создадим документ Microsoft *Word* и добавим на него следующие кнопки [табл. 6.3](https://www.intuit.ru/studies/courses/494/350/lecture/8361?page=1#table.6.3).

|  |  |  |
| --- | --- | --- |
| Таблица 6.3. Кнопки на листе | | |
| **Имя кнопки** | **Надпись** | **Номер листинга** |
| cmd\_Len | Длина строки | [листинг 6.1](https://www.intuit.ru/studies/courses/494/350/lecture/8361?page=1#example.6.2). |
| cmd\_Conv | Конверсия | [листинг 6.2](https://www.intuit.ru/studies/courses/494/350/lecture/8361?page=1#example.6.3). |
| cmd\_Mid | Вырезание | [листинг 6.3](https://www.intuit.ru/studies/courses/494/350/lecture/8361?page=1#example.6.4). |
| cmd\_Asc | Коды | [листинг 6.5](https://www.intuit.ru/studies/courses/494/350/lecture/8361?page=1#example.6.5). |

Обработчик нажатия кнопки cmd\_Len ([листинг 6.2](https://www.intuit.ru/studies/courses/494/350/lecture/8361?page=1#example.6.2).) будет содержать решение следующей задачи: сообщить пользователю длину введенного текста. Очевидно, для решения этой задачи нам понадобится *функция* Len.

'Переменная для хранения входной строки

Dim str\_InpStr As String

'Переменная для хранения найденной длины строки

Dim lng\_StrLen As Long

str\_InpStr = InputBox("Введите строку")

'Вычисляем длину строки

lng\_StrLen = Len(str\_InpStr)

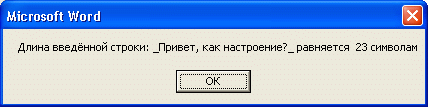
MsgBox ("Длина введенной строки: \_" + \_

str\_InpStr + "\_ равняется " + Str(lng\_StrLen) + \_

" символам")

Листинг 6.2. Пример использования функции Len

На [рис. 6.2](https://www.intuit.ru/studies/courses/494/350/lecture/8361?page=1#image.6.2). вы можете видеть результат вычисления длины строки.



**Рис. 6.2.**Результат вычисления длины строки

Теперь займемся *конверсией* символов - функциями LCase и UCase (листинг 6. 3.).

'Переменная для хранения входной строки

Dim str\_InpStr As String

'Переменная для хранения измененной строки

Dim str\_NewStr As String

str\_InpStr = InputBox("Введите текст")

'В str\_NewStr окажется введеная строка

'в которой все прописные буквы заменены строчными

str\_NewStr = LCase(str\_InpStr)

MsgBox ("Измененная строка: " + str\_NewStr)

'Теперь в str\_NewStr будет та же строка

'в которой все буквы стали прописными

str\_NewStr = UCase(str\_InpStr)

MsgBox ("Измененная строка: " + str\_NewStr)

Листинг 6.3. Пример использования функций LCase и UCase

На очереди - вырезание символов - функции Mid, LTrim, Rtrim, Left, Right ([листинг 6.4](https://www.intuit.ru/studies/courses/494/350/lecture/8361?page=1#example.6.4).) Среди этих функций наиболее мощной является Mid - используя ее, можно делать со строками очень много всего.

'Переменная для хранения входной строки

Dim str\_InpStr As String

'Переменная для хранения вырезанных символов

Dim str\_NewStr As String

'Зададим строку, с которой удобно будет работать

str\_InpStr = " Здравствуйте, Александр "

'Функции удаления пробелов

'При выводе строки ставим перед ее началом

'и концом символ "\_" для того чтобы

'лучше было видно наличие

'или отсутствие пробелов

MsgBox ("Мы работаем с такой строкой: " + \_

"\_" + str\_InpStr + "\_")

'LTrim - присваиваем результаты работы

'переменной и выводим информацию в MsgBox

str\_NewStr = LTrim(str\_InpStr)

MsgBox ("Результат работы LTrim: " + \_

"\_" + str\_NewStr + "\_")

'RTrim

MsgBox ("Результат работы RTrim: " + \_

"\_" + RTrim(str\_InpStr) + "\_")

'Trim

MsgBox ("Результат работы Trim: " + \_

"\_" + Trim(str\_InpStr) + "\_")

'Left - вырезаем из строки str\_InpStr 12

'символов предварительно очистив ее

'от начальных пробелов

str\_NewStr = Left(LTrim(str\_InpStr), 12)

MsgBox ("Первые 12 символов слева: " + \_

str\_NewStr)

'Right - аналогично Вырезаем 9 символов справа

str\_NewStr = Right(RTrim(str\_InpStr), 9)

MsgBox ("Первые 9 символов справа: " + \_

str\_NewStr)

'Функция Mid - для начала выведем по одному символу

'со второй и пятнадцатой позиции строки

'преварительно очищенной от лишних пробелов

'в начале и в конце

str\_NewStr = Mid(Trim(str\_InpStr), 2, 1)

MsgBox ("Второй символ введенной строки: " + \_

str\_NewStr)

str\_NewStr = Mid(Trim(str\_InpStr), 15, 1)

MsgBox ("Пятнадцатый символ введенной строки: " + \_

str\_NewStr)

'Выведем 5 символов, начиная с 15 символа

str\_NewStr = Mid(Trim(str\_InpStr), 15, 5)

MsgBox ("Пять символов строки с 15-й позиции: " + \_

str\_NewStr)

Листинг 6.4. Пример использования функций для вырезания символов

Теперь рассмотрим примеры работы функций *Asc*, *Chr* и функции String. Чтобы работать с функциями *Asc* и *Chr* нам нужно познакомиться с понятием таблицы символов *ASCII*.

*ASCII* расшифровывается как American *Standard* Code For *Information* Intercharge - американский стандартный код для обмена информацией. Каждый символ в *ASCII* закодирован восьмибитным кодом. В результате получается *таблица*, в которой каждому управляющему символу, цифре, букве латинского или национального алфавитов сопоставлен свой код. Коды записывают в различных представлениях - в основном - в шестнадцатеричном и десятичном. Мы будем пользоваться десятичной записью.

Мы не будем приводить здесь таблицу *ASCII* полностью, приведем лишь некоторые полезные коды и диапазоны кодов.

Коды в диапазоне 0-31 имеют *управляющие* символы. Символ возврата каретки (тот самый, который вставляется в документ при нажатии клавиши **Enter** ) имеет код 13.

Коды в диапазоне 32-127 имеют латинские символы, цифры, знаки препинания - эта часть таблицы остается постоянной для различных кодовых таблиц. Например, *пробел* имеет код 32, точка - 46. *Диапазон* 48-57 занимают цифры от 0 до 9, *диапазон* 65-90 занимают заглавные латинские буквы от A до Z, *диапазон* 97-122 - строчными буквы a-z.

В диапазоне 128-225 расположены символы национальной кодировки. В русифицированных версиях MS *Windows* обычно применяется кодовая страница MS *Windows* 1251. В ней коды 192-223 имеют заглавные буквы от А до Я, 224-255 - строчные буквы от а до я.

Используя коды символов можно вводить в документы (или записывать в файлы, создаваемые программно) символы, которые нельзя ввести с клавиатуры.

В коде обработчика Click для cmd\_Asc создадим программу ([листинг 6.5](https://www.intuit.ru/studies/courses/494/350/lecture/8361?page=1#example.6.5).), которая сначала запрашивает у пользователя ввод символа, после чего выводит его *ASCII* код, потом - ввод кода, после чего выводит соответствующий ему символ. Так же здесь мы посмотрим на то, как работает *функция* String:

'Переменная для хранения кода символа

Dim num\_AscNumber

'Переменная для хранения символа

Dim str\_Char

'Начало блока перевода кода в символ

num\_AscNumber = Val(InputBox("Введите код символа"))

'В переменной str\_Char теперь хранится символ

'с кодом num\_AscNumber

str\_Char = Chr(num\_AscNumber)

MsgBox (str\_Char + " - символ с кодом " + \_

Str(num\_AscNumber))

'Начало блока перевода символа в код

str\_Char = InputBox("Введите символ")

'Теперь в переменной num\_AscNumber хранится

'Код символа, введенного в str\_Char

num\_AscNumber = Asc(str\_Char)

MsgBox ("Символу " + str\_Char + " соответствует код" + \_

Str(num\_AscNumber))

'А теперь - пример функции String.

'выведем в окне сообщения 15 символов \*

MsgBox ("15 символов \*: " + String(15, "\*"))

Листинг 6.5. Пример использования функций Chr, Asc, String

**6.3. Пользовательские процедуры и функции**

[06-03-Пользовательские процедуры и функции.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/6/files/06_03.docm)- пример к п. 6.3. - 6.5.

Выше мы пользовались *встроенными* процедурами и функциями *VBA*. Может показаться, что их достаточно много для решения любой задачи. Но даже при таком разнообразии порой нужны совершенно особые процедуры, которые *пользователь* может создать самостоятельно.

Напомним, что главное отличие процедуры от функции заключается в том, что *функция* возвращает в точку вызова некое *значение*, которое, как правило, является результатом обработки переданной функции информации. А процедура лишь выполняет какие-либо действия, но ничего в точку вызова не возвращает.

Например, очевидно, что при обработке такой последовательности команд: num\_A = num\_B + val(str\_C) вместо выражения val(str\_C) подставляется числовое *значение* переменной str\_C, найденное благодаря функции Val. Или, скажем, такое *выражение*: num\_A = InputBox("Введите число"). Здесь *функция* InputBox возвращает введенное пользователем число в переменную num\_A.

Процедуры, как уже было сказано, лишь выполняют какие-то действия, не возвращая никаких значений в точку вызова. Например, если мы вызываем окно сообщения с надписью, не настраивая никаких дополнительных параметров, то мы можем решить, что имеем дело с процедурой. Такой вызов: MsgBox ("Привет"), будет отлично работать. В то же время, MsgBox может вести себя как *функция*, возвращая в точку вызова код нажатой кнопки при настройках кнопок окна сообщения отличных от стандартных.

Разобравшись в отличиях процедур и функций, поговорим о том, зачем они нам нужны. Во-первых, процедуры удобно использовать для сокращения объема программы, выделяя в них часто используемые *блоки операторов*. Если вам придется создавать реальные программы, вы очень скоро убедитесь в том, что некоторые *операции* (например - *запись* каких-либо значений на *лист* MS *Excel*, создание и сохранение документов и т.д.) могут быть нужными в различных местах программы и занимать порой несколько десятков строчек кода. Нерационально каждый раз копировать эти участки кода в нужное *место*. Во-первых - увеличивается размер текста программы, во-вторых - если вам понадобится изменить что-нибудь в часто используемом наборе операторов - придется искать все такие участки и редактировать каждый из них. Гораздо удобнее будет выделить часто используемый участок в виде самостоятельной процедуры и каждый раз, когда он будет нужен, вызывать его с помощью одной лишь строчки кода.

**6.4. Пользовательская процедура**

Рассмотрим пример создания и использования *пользовательской процедуры*. Предположим, нам нужно опросить пятерых пользователей, обработав и записав их ответы в *файл*. Самое главное в этом примере не то, куда мы будем записывать ответы, и как их будем обрабатывать, а то, что блок кода будет повторяться для каждого пользователя. Сначала напишем этот блок в расчете на одного пользователя. Для этого добавим в документ Microsoft *Word* кнопку cmd\_User\_Data и создадим для нее такой обработчик события ([листинг 6.6](https://www.intuit.ru/studies/courses/494/350/lecture/8361?page=2#example.6.6).):

Private Sub cmd\_User\_Data\_Click()

Dim str\_Name As String

Dim byte\_Age As Byte

MsgBox ("Здравствуйте, вы пользователь № 1")

str\_Name = InputBox("Введите ваше имя")

byte\_Age = InputBox("Введите ваш возраст")

'Здесь находится блок обработки и

'записи введенных данных

End Sub

Листинг 6.6. Опрос пользователя - заготовка для процедуры

Для наглядности мы оставили *операторы* начала и конца процедуры.

Давайте подумаем, что нужно сделать с кодом при переходе ко второму пользователю. Очевидно, что мы сможем работать с теми же переменными, что и в первом случае, будем использовать те же *операторы*. Единственным, что изменится, будет надпись в первом окне сообщения. Вместо "Здравствуйте, вы *пользователь* №1" будет выведено "Здравствуйте, вы *пользователь* №2".

Это значит, что мы можем использовать весь код, который находится в данный момент в обработчике, для создания собственной процедуры.

Чтобы создать процедуру, добавим в проект новый *модуль*, например, командой **Insert o Module** (Вставить o *Модуль*).

В модуле создадим для начала "скелет" процедуры. Он будет выглядеть так ([листинг 6.7](https://www.intuit.ru/studies/courses/494/350/lecture/8361?page=2#example.6.7).):

Public Sub UserInput(UserNumber As Integer)

'пользовательская процедура

'для ввода и обработки данных

End Sub

Листинг 6.7. Начало и конец текста пользовательской процедуры

Внутри процедуры - там, где будет располагаться ее тело, мы разместили комментарии. Вы уже знаете, что End Sub означает конец процедуры. А вот первая строка объявления выглядит гораздо интереснее. Давайте разберем ее составные части.

UserInput - это имя процедуры. Мы выбираем его самостоятельно. В скобках после имени находится *объявление переменной*, которую можно передать процедуре в качестве параметра. В нашем случае это *переменная*, имеющая имя UserNumber и тип Integer. Вспомните, как мы работаем с функцией MsgBox - вызывая ее мы пишем такой код: MsgBox ("Привет"). В данном случае мы передаем функции MsgBox текстовый *параметр* "Привет", который она выведет на экран. Точно так же, мы сможем передать нашей процедуре числовой *параметр* типа Integer, который будет "виден" ей под именем UserNumber. Вызов нашей процедуры для первого пользователя из обработчика события Click будет выглядеть так:

UserInput (1)

Тот номер, который мы указали в скобках, будет присвоен переменной UserNumber и мы сможем пользоваться ей внутри процедуры. Давайте проиллюстрируем это. Как вы помните, выше мы определили, что в общем коде будет изменяться лишь номер пользователя. Поэтому перепишем строку приглашения таким образом, чтобы для каждого номера пользователя она выводила бы собственный текст. После этого текст процедуры будет выглядеть так ([листинг 6.8](https://www.intuit.ru/studies/courses/494/350/lecture/8361?page=2#example.6.8).):

Public Sub UserInput(UserNumber As Integer)

MsgBox \_

("Здравствуйте, вы пользователь № " + \_

Str(UserNumber))

End Sub

Листинг 6.8. Модификация пользовательской процедуры

Теперь рассмотрим остальные части первой строки процедуры. *Ключевое слово* Sub означает, что перед нами процедура. А модификатор доступа Public означает, что эту процедуру можно вызвать из любого модуля проекта. Существуют и другие *модификаторы доступа*, в частности, Private. Он дает *доступ* к процедуре только из того модуля, где она объявлена.

Теперь доработаем процедуру и обработчик события так, чтобы с их помощью мы могли решить поставленную выше задачу. Ниже вы можете видеть код обработчика события Click для кнопки ([листинг 6.9](https://www.intuit.ru/studies/courses/494/350/lecture/8361?page=2#example.6.9).) и процедуры ([листинг 6.10](https://www.intuit.ru/studies/courses/494/350/lecture/8361?page=2#example.6.10).).

Private Sub cmd\_User\_Data\_Click()

UserInput (1)

UserInput (2)

UserInput (3)

UserInput (4)

UserInput (5)

End Sub

Листинг 6.9. Код события Click для cmd\_User\_Data

Public Sub UserInput(UserNumber As Integer)

Dim str\_Name As String

Dim byte\_Age As Byte

MsgBox \_

("Здравствуйте, вы пользователь № " + \_

Str(UserNumber))

str\_Name = InputBox("Введите ваше имя")

byte\_Age = InputBox("Введите ваш возраст")

End Sub

Листинг 6.10. Окончательный вариант пользовательской процедуры

Теперь поговорим о функциях.

**6.5. Пользовательская функция**

Рассмотрим пример создания и использования функции, которая выполняет очень простую задачу: возводит переданное ей число во вторую степень.

*Функция*, реализующая эту задачу, выглядит так ([листинг 6.11](https://www.intuit.ru/studies/courses/494/350/lecture/8361?page=2#example.6.11).):

Public Function Square(num\_One As Double) As Double

Square = num\_One ^ 2

End Function

Листинг 6.11. Пользовательская функция

*Объявление функции* очень похоже на объявление процедуры. Здесь мы используем модификатор доступности Public, дающий *доступ* к функции из всех модулей. *Ключевое слово* Function означает, что мы объявляем функцию. Следом за ним, в скобках, идет объявление параметров, которые мы можем передать функции. В нашем случае это - одна *переменная* num\_One типа Double. После объявления имени и параметров функции следует объявление типа самой функции, или, точнее - типа возвращаемого функцией значения. В нашем случае с помощью конструкции As Double мы задали тип функции Double. При объявлении функции можно не указывать тип - тогда он автоматически будет установлен в Variant.

Последняя строка *объявления функции* - End Function, означает конец функции. Как и в случае с процедурой, внутри тела функции можно разместить *операторы*, необходимые для выполнения задачи. Однако, здесь есть одна очень важная особенность. После того, как найдено *значение*, которое *функция* должна возвратить, в тексте кода функции нужно присвоить это *значение* переменной, которая имеет то же имя, что и *функция*. В нашем случае, при имени функции Square это присвоение выглядит как Square = num\_One ^ 2.

Для испытания функции добавим в документ еще одну кнопку - cmd\_UserCalc, подписанную как *Функция* с таким кодом ([листинг 6.12](https://www.intuit.ru/studies/courses/494/350/lecture/8361?page=2#example.6.12).):

Private Sub cmd\_UserCalc\_Click()

Dim num\_Res As Double

Dim num\_Input As Double

num\_Input = CDbl(InputBox("Введите число"))

num\_Res = Square(num\_Input)

MsgBox (Str(num\_Input) + " во 2-й степени = " + \_

Str(num\_Res))

End Sub

Листинг 6.12. Работа с пользовательской функцией

Как видите, здесь мы работаем с *пользовательской функцией* точно так же, как работали бы с одной из *встроенных функций*. Для наглядности мы использовали переменную, в которую записываем *значение*, возвращаемое функцией.

**6.6. Пользовательские типы данных**

[06-04-Пользовательские типы данных.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/6/files/06_04.docm)- пример к п. 6.6.

Иногда возникает необходимость в работе с данными, имеющими особенную структуру. Например, какой тип должна иметь *переменная*, которая может хранить код пользователя, его ФИО, возраст, *адрес*, телефон, дату рождения? Очевидно, что *встроенными типами данных* тут не обойтись. Можно, конечно, представить себе строковую переменную, которая содержит все эти данные, записанные подряд, но работать с такой переменной будет очень неудобно. В подобных случаях на помощь приходят *пользовательские типы данных*. Для определения нового типа данных используется конструкция Type - End Type.

Рассмотрим пример. Создадим документ Microsoft *Word*, добавим с помощью редактора *Visual* *Basic* новый *модуль* и разместим в нем такой код ([листинг 6.13](https://www.intuit.ru/studies/courses/494/350/lecture/8361?page=3#example.6.13).):

Type Worker ' начало объявления типа данных

Usercode As Integer

Name As String

Phone As String

BirthDate As Date

End Type

Public Sub WorkWithUser()

Dim wrk\_NewUser As Worker

Dim wrk\_TestUser As Worker

Dim str\_UserName As String

wrk\_NewUser.Usercode = 1

wrk\_NewUser.Name = "Петров Петр Петрович"

wrk\_NewUser.Phone = "8(928)8888888"

wrk\_NewUser.BirthDate = #8/12/1980#

'Обмен данными между переменными

'пользовательского типа

'адекватен обычному обмену

wrk\_TestUser = wrk\_NewUser

'Присвоим строковой переменной

'значение одной из частей

'пользовательской переменной

str\_UserName = wrk\_TestUser.Name

MsgBox ("Имя пользователя: " + str\_UserName)

End Sub

Листинг 6.13. Код модуля с объявлением пользовательского типа данных

Новый *тип данных* должен быть объявлен вне процедуры - на уровне модуля. Мы специально привели здесь, вместе с текстом объявления типа, процедуру, в которой используется этот тип. Как видите, внутри объявления типа (между Type и End Type ), находятся объявления переменных, а сразу после ключевого слова Type следует имя типа. Мы дали типу данных имя *Worker*, а это значит, что объявляя переменную в модуле, мы будем использовать это имя для указания ее типа данных. Имя нашего типа данных используется при объявлении переменной точно так же, как и имена встроенных типов - String, Byte и т.д. Мы объявили три переменные. wrk\_NewUser типа *Worker*, wrk\_TestUser того же типа и str\_UserName - обычную строковую переменную.

После того, как *переменная* типа *Worker* объявлена, мы можем работать с ней. А именно, обращение к элементам переменной ведется через точку с использованием имен "внутренних" переменных типа. Сначала мы по очереди присваиваем каждой части переменной wrk\_NewUser соответствующие значения. Далее мы присваиваем *значение* переменной wrk\_NewUser переменной wrk\_TestUser. Как видите, работа с пользовательскими переменными одинакового типа ни чем не отличается от работы с обычными переменными. Несмотря на сложную структуру переменной, обычный *оператор присваивания* отлично справляется с переносом значений из одной переменной в другую. Следующий этап нашей программы - извлечение значения одной из частей переменной типа *Worker*, а конкретно - имени пользователя, в строковую переменную str\_UserName. После этого мы выводим переменную str\_UserName в окне сообщения.

**6.7. Константы**

[06-05-Константы.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/6/files/06_05.docm)- пример к п. 6.7.

*Константы* очень похожи на переменные. Они имеют имена, которые строятся по тем же правилам, в них могут храниться различные значения и т.д. Главное отличие *констант*от переменных в том, что *константы* нельзя изменять в ходе выполнения программы. Это удобно в тех случаях, когда вы собираетесь много раз использовать в программе какое-то *значение* и хотите, чтобы оно гарантированно оставалось неизменным.

Для объявления *констант* используется *ключевое слово* Const. При объявлении *константы* нужно обязательно указать ее имя и присвоить ей нужное *значение*. При необходимости можно указать и тип *константы*. Вот как выглядит работа с *константами* в процедуре ([листинг 5.24](https://www.intuit.ru/studies/courses/494/350/lecture/8361?page=3#example.5.24).):

Const str\_Name As String = "Александр"

Const int\_Size As Double = 18000

Dim num\_NewSize As Double

MsgBox ("Здравствуйте, " + str\_Name)

num\_NewSize = int\_Size \* 2

Листинг 5.24. Работа с константами

Здесь мы объявили пару *констант* - одну строковую для хранения имени пользователя, вторую - типа Double. Строковую константу мы используем для построения строки вывода в операторе MsgBox, а числовую применяем для построения выражения, результат вычислении которого записывается в переменную num\_NewSize.

В *VBA* имеется обширный набор встроенных *констант*. Они используются для работы с цветом (например, vbRed - красный и т.д.) для задания типов окон сообщений и во многих других случаях. Как правило, *константы*, допустимые в том или ином случае, можно найти в справочном материале, который появляется при наборе команд. Так же, подробные сведения о встроенных *константах* содержатся в справочной системе *VBA*.

**6.8. Выводы**

В этой лекции мы завершили обсуждение основ *VBA* - теперь вы можете писать простые программы. Но мы еще не разговаривали о более продвинутых возможностях программирования. О них - в следующей лекции.

Лекция 7. Массивы, циклы, принятие решений

**7.1. Массивы**

Решим простую задачу: ввести в ответ на вопросы программы фамилии сотрудников. Если фамилий немного - 5 или 10 - использование переменных, с которыми вы уже знакомы, вполне оправдано. Программа будет состоять из нескольких строк такого вида:

a = InputBox("Введите фамилию сотрудника").

Как быть, если нужно работать со списком из 20 фамилий? А если их будет 50, 1000 или их количество должно быть определено в ходе выполнения программы?

Для обработки больших объемов информации использовать переменные неудобно. Что же делать? Ответ прост: использовать *массивы*.

Массив - это именованный набор индексированных ячеек. Ячейки так же называют элементами или индексированными переменными.

У каждого *массива* 5 основных характеристик: имя, размерность, число элементов, номер первого элемента и тип элементов.

* **Имя**- правила именования *массивов* аналогичны правилам *именования переменных*.
* **Размерность**- одномерные *массивы* напоминают одну строку таблицы, каждая ячейка которой содержит какие-то данные. Многомерные *массивы* имеют больше измерений. Их можно сравнивать с таблицами, имеющими множество строк и столбцов и с наборами таблиц.
* **Нумерация элементов**подчиняется следующим правилам:
  + По умолчанию нумерация элементов *массива* начинается с 0. Первый по счету элемент получит индекс 0, второй - 1 и т.д.
  + В объявлении отдельного *массива* можно явно указать индекс его первого и последнего элемента, разделив их ключевым словом To.
  + Если вы хотите, чтобы индексы всех *массивов* начинались с 1, добавьте в раздел объявлений модуля (вне процедур, функций и обработчиков событий) команду Option Base 1.
* **Тип**- подчиняется тем же правилам, которые мы ранее рассмотрели для переменных. Если пользователь не указал его явно, массив получит тип по умолчанию - Variant. Это требует больше системных ресурсов, но позволяет обрабатывать значения различных типов.

Не всегда количество элементов и размерность *массива* известны до начала работы программы. VBA умеет работать с *динамическими массивами*, параметры которых можно менять в ходе выполнения программы. О них мы поговорим ниже.

**7.2. Одномерные массивы**

[07-01-Одномерный массив.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/7/files/07_01.docm)- пример к п. 7.2.

Для объявления *массивов* используют оператор Dim. Объявить массив можно двумя способами. Первый заключается в указании общего количества элементов. Например, так:

Dim MyArrayA(30) As Single

Объявленный массив MyArrayА содержит 31 элемент (с индексами от 0 до 30) типа Single. Поскольку нумерация явно не задана, элементы получат индексы по обычным правилам.

Можно объявить массив и другим способом:

Dim MyArrayB(1 To 25)

Массив MyArrayB содержит 25 элементов. Границы нумерации заданы явно - первый элемент получит индекс 1, второй - 2 и т.д. Тип не указан - в *массиве* можно хранить любые данные.

Работа с элементами *массива* ничем не отличается от работы с переменными. Давайте решим следующую задачу:

1. Объявить одномерный массив на 3 элемента
2. Внести в первый элемент число 5 в программе, во второй - запросив значение с помощью окна ввода
3. Вычислить в третьем элементе *массива* произведение значений, хранящихся в первом и втором элементах.
4. Вывести полученное значение в окне сообщения.

Решение задачи приведено в [листинге 7.1](https://www.intuit.ru/studies/courses/494/350/lecture/8363?page=1#example.7.1).

Dim A(2)

A(0) = 5

A(1) = InputBox("Введите значение второго элемента")

A(2) = A(0) \* A(1)

MsgBox A(2)

Листинг 7.1. Работа с массивом

В конце работы программы, если на вопрос о вводе числа мы введем число 2, массив A будет иметь такой вид - [табл. 7.1](https://www.intuit.ru/studies/courses/494/350/lecture/8363?page=1#table.7.1).

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица 7.1. Массив А после работы программы | | | |
| **Индекс** | **0** | **1** | **2** |
| Значение | 5 | 2 | 10 |

Как видите, серьезное отличие *массивов* от переменных - использование индексов для работы с различными элементами *массива*. Однако, можно заметить, что эта задача легко решается с использованием переменных, без *массивов*.

Возможно, сейчас вы еще не вполне осознаете полезность *массивов*. Если сейчас вы напишите программу для ручного ввода данных в несколько элементов *массива*, она получится такой же громоздкой, как раньше. Так чем же *массивы* лучше? Ответ на этот вопрос кроется в использовании циклов.

**7.3. Циклы**

Циклы позволяют выполнять одни и те же команды много раз. В [табл. 7.2](https://www.intuit.ru/studies/courses/494/350/lecture/8363?page=1#table.7.2). собрана информация об основных видах циклов.

|  |  |
| --- | --- |
| Таблица 7.2. Виды циклов | |
| **Название цикла** | **Вид** |
| For - Next | С фиксированным количеством повторов. Выполняется заданное количество раз |
| While - Wend | С предусловием. Если не верно условие, заданное на входе в цикл, может не выполниться ни разу |
| Do - Loop | С постусловием. Выполняется по меньшей мере один раз. |

**7.4. Цикл For - Next**

[07-02-Цикл For-Next.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/7/files/07_02.docm)- пример к п. 7.4.

Решим задачу: вывести цифры от 1 до 10 в окнах сообщений. Ее можно решить и без использования циклов, написав 10 строк такого вида: MsgBox ("1"). А вот конструкция For - Next позволяет делать то же самое гораздо изящнее. В листинге 7.2. вы можете найти решение задачи.

For i = 1 To 10

MsgBox (i)

Next i

Листинг 7.2. Вывод цифр от 1 до 10 в цикле

Ключевое слово For задает начало цикла. После него следует переменная i, которая увеличивается при каждом проходе цикла. В ходе работы конструкции For - Next значение i изменяется от 1 до 10.

Начальное значение счетчика устанавливается при входе в цикл. Мы просто приравниваем i какому-либо числу. Конечное значение переменной задается после ключевого слова To. Переменная i доступна внутри цикла - ведь именно ее мы выводим в окне сообщения.

Ключевое слово Next с указанием переменной, к которой оно относится, закрывает цикл.

В качестве первого и последнего значения счетчика цикла можно использовать какую-нибудь переменную. Она может быть определена в ходе выполнения программы, но до входа в конструкцию For - Next.

Вы могли заметить, что переменная цикла меняется с приращением 1. Это приращение можно задать в явном виде с помощью ключевого слова Step. В листинге 7.3. вы можете видеть пример оператора For - Next, выводящего нечетные числа в диапазоне от 1 до 10.

For i = 1 To 10 Step 2

MsgBox (i)

Next i

Листинг 7.3. Вывод нечетных цифр в диапазоне от 1 до 10 в цикле

С помощью ключевого слова Step можно создать не только возрастающий, но и убывающий счетчик. Для этого в Step надо указать отрицательное число и проследить за тем, чтобы первое значение переменной цикла было больше последнего. Например, так: For i=10 to 1 Step -1. Первое значение переменной в таком цикле будет равняться 10, второе - 9 и т.д. - до 1.

Теперь перейдем к совместному использованию циклов и *массивов*. Напишем программу, которая предлагает пользователю ввести 10 фамилий, сохраняет их в *массиве*, а потом выводит в окнах сообщений.

Готовая программа приведена в листинге 7.4.

Dim MyArray(9)

For i = 0 To 9

MyArray(i) = InputBox("Введите фамилию №" & i + 1)

Next i

For i = 0 To 9 'Начало еще одного цикла

MsgBox ("Фамилия №" & (i + 1) & " " & MyArray(i))

Next i 'Конец цикла

Листинг 7.4. Ввод и вывод 10 фамилий

Обсудив цикл типа For-Next, продолжим разговор о *массивах*. На очереди - многомерные *массивы*.

**7.5. Многомерные массивы**

[07-03-Многомерные массивы.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/7/files/07_03.docm)- пример к п. 7.5.

Многомерные *массивы* имеют несколько измерений. Чаще всего применяются двумерные *массивы* (матрицы). Матрицу можно представить в виде обычной таблицы с несколькими строками и столбцами.

Для того чтобы объявить двумерный массив, нужно воспользоваться командой Dim с указанием размерности каждого из измерений. Остальные правила объявления таких *массивов* и работы с ними аналогичны таковым для одномерных *массивов*. Например, мы можем указать лишь размеры измерения *массива*:

Dim MyArrayA(10, 1) As Single

Массив MyArrayА содержит 11 строк и 2 столбца типа Single.

Можно в явном виде задать границы размерностей:

Dim MyArrayB(1 To 25, 1 To 5)

Массив MyArrayB содержит 25 строк и 5 столбцов. Границы нумерации заданы явно. Тип не указан - в *массиве* можно хранить любые данные.

В листинге 7.5. приведен пример программы, которая объявляет двумерный массив 5х2 и предлагает ввести в него фамилии и номера телефонов сотрудников.

Dim MyArray(1 To 5, 1 To 2)

For i = 1 To 5

MyArray(i, 1) = InputBox("Введите фамилию №" & i)

MyArray(i, 2) = InputBox("Введите Телефон №" & i)

Next i

Листинг 7.5. Работа с массивом размерностью 5х2

В [табл. 7.3](https://www.intuit.ru/studies/courses/494/350/lecture/8363?page=1#table.7.3). вы можете видеть массив MyArray после заполнения его фамилиями и номерами телефонов. В MyArray (1,1) мы внесли фамилию "Иванов", в MyArray (1,2) - телефон Иванова 898989898 и т.д.

|  |  |  |
| --- | --- | --- |
| Таблица 7.3. Заполненный массив MyArray | | |
| **Индекс** | **1** | **2** |
| 1 | Иванов | 898989898 |
| 2 | Петров | 343434343 |
| 3 | Сидоров | 565656565 |
| 4 | Александров | 121111212 |
| 5 | Маринин | 545454544 |

Эта программа очень похожа на те, которые мы писали для работы с одномерными *массивами*. В цикле, тело которого повторяется 5 раз, мы поочередно запрашиваем фамилию и номер телефона.

Один цикл неудобно использовать для работы с *массивами* больших размерностей. Нетрудно представить себе, какой громоздкой получится решение задачи копирования одной матрицы 100х100 в другую такую же. Кстати, при обработке данных в Microsoft Excel вам постоянно придется обращаться с большими двумерными матрицами. К счастью, существует механизм вложенных циклов, который помогает решать подобные задачи.

**7.6. Вложенные циклы For-Next**

[07-04-Вложенные циклы.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/7/files/07_04.docm) - пример к п. 7.6.

Принцип работы вложенных циклов кроется в их названии. Все очень просто - один цикл вкладывается в другой. Например, для заполнения *массива* 10х10 случайными целыми числами от 1 до 10 можно написать такую программу ([листинг 7.6](https://www.intuit.ru/studies/courses/494/350/lecture/8363?page=1#example.7.6).)

Dim MyArray(1 To 10, 1 To 10)

For i = 1 To 10

For j = 1 To 10

MyArray(i, j) = Int(Rnd(1) \* 10)

Next j

Next i

Листинг 7.6. Работа с массивом 10х10

Внешний цикл ( i ) выполняется один раз, после чего внутренний ( j ) - десять раз. За один проход внешнего цикла внутренний выполняет десять - заполняется первая строка *массива* (с индексами от 1,1 до 1,10) и т.д.

**7.7. Динамические массивы**

[07-05-Динамические массивы.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/7/files/07_05.docm)- пример к п. 7.7.

Когда вы используете массив, не всегда известно заранее, сколько элементов он будет иметь. В такой ситуации можно объявить массив, который содержит заведомо больше элементов, чем может понадобиться, но это приведет к нерациональному использованию системных ресурсов. Что же делать? Динамические *массивы* - вот достойный ответ на этот вопрос.

Решим задачу. Программа просит пользователя ввести количество сотрудников, которое сохраняет в переменной ArraySize, а потом создает массив, одна из размерностей которого равняется ArraySize.

Чтобы воспользоваться *динамическим массивом*, сначала нужно объявить пустой массив, например, командой Dim MyArray(), а потом задать размерность *массива* командой ReDim (листинге 7.7.)

Dim MyArray()

ArraySize = InputBox("Введите количество сотрудников")

ReDim MyArray(1 To ArraySize, 1 To 2)

Листинг 7.7. Работа с динамическим массивом

В итоге, если на вопрос программы о количестве сотрудников мы ввели число 15, будет создан двумерный массив размерностью 15х2.

Если в программе возникла ситуация, когда последней размерности объявленного и заполненного *массива* не хватает для хранения данных, вы можете увеличить его командой ReDim с ключевым словом *Preserve*. Благодаря ему данные, внесенные ранее в массив, будут сохранены. Например, для добавления двух дополнительных столбцов в *динамический массив* из [листинга 7.7](https://www.intuit.ru/studies/courses/494/350/lecture/8363?page=2#example.7.7). нужно использовать такую команду:

ReDim Preserve MyArray(1 To ArraySize, 1 To 4)

**7.8. Дополнительные команды работы с массивами**

[07-06-Дополнительные команды.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/7/files/07_06.docm)- пример к п. 7.8.

Для работы с *массивами* вы можете использовать еще некоторые команды.

Array (*Список аргументов*)- позволяет быстро заполнять массив. Например, в листинге 7.8. массив MyArray заполняется числами 1, 2, 6, 9 и 19, после чего первый элемент *массива* выводится в окне сообщения.

Dim MyArray

MyArray = Array(1, 2, 6, 9, 19)

MsgBox MyArray(0)

Листинг 7.8. Работа с оператором Array

IsArray (Имя переменной) - возвращает True если переменная является массивом. Например, в листинге 7.9. мы объявляем две переменные - одну из них как массив, вторую - как обычную переменную. Далее мы используем оператор IsArray для проверки того, является ли переменная массивом. После чего программа выводит соответствующее сообщение. Здесь мы использовали оператор сравнения If, подробности о котором мы рассмотрим ниже.

Dim MyArray(10)

Dim MyArr

If IsArray(MyArray) Then \_

MsgBox ("Переменная MyArray - массив") \_

Else MsgBox ("Переменная MyArray - не массив")

If IsArray(MyArr) Then \_

MsgBox ("Переменная MyArr - массив") \_

Else MsgBox ("Переменна MyArr - не массив")

Листинг 7.9. Работа с функцией IsArray

LBound (Имя Массива, Размерность) - возвращает нижнюю границу для указанной *размерности массива*.

UBound (Имя Массива, Размерность) - возвращает верхнюю границу для указанной *размерности массива*.

Рассмотрим пример. Создадим *динамический двумерный массив*, размерности которого заданы с помощью генератора случайных чисел. После этого с помощью операторов LBound и UBound узнаем размерности *массива* и выведем их в окнах сообщений. Далее - используем двойной цикл для заполнения *массива* случайными числами ([листинг 7.10](https://www.intuit.ru/studies/courses/494/350/lecture/8363?page=2#example.7.10).)

Dim MyArray()

ReDim MyArray(Int(Rnd \* 5 + 5), Int(Rnd \* 5 + 5))

MsgBox ("Двумерный массив MyArray:" + Chr(13) + \_

"Первая размерность:" + \_

Str(LBound(MyArray, 1)) + " -" + \_

Str(UBound(MyArray, 1)) + Chr(13) + \_

"Вторая размерность:" + \_

Str(LBound(MyArray, 2)) + " -" + \_

Str(UBound(MyArray, 2)))

For i = LBound(MyArray, 1) To UBound(MyArray, 1)

For j = LBound(MyArray, 2) To UBound(MyArray, 2)

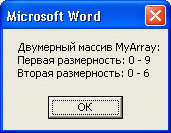
MyArray(i, j) = Int(Rnd \* 100)

Next j

Next i

Листинг 7.10. Работа с функциями LBound и UBound

В нашем случае команда LBound для обеих размерностей *массива* возвращает 0 так как по умолчанию нумерация элементов *массива* начинается с 0. А вот Ubound возвращает границу каждой из размерностей, которая установлена случайным образом с помощью оператора ReDim. На [рис. 7.1](https://www.intuit.ru/studies/courses/494/350/lecture/8363?page=2#image.7.1). вы можете видеть окно сообщения c информацией о границах *массива*.



**Рис. 7.1.**Сообщение о размерностях массива

Erase Имя\_массива - очистить массив. Элементы обычных *массивов*, содержащих числовые данные, обнуляются. Если мы применим команду Erase к *массиву* строк - каждый его элемент будет хранить строку нулевой длины (""). Применяя команду Erase к *динамическому массиву*, мы очищаем память, выделенную этому *массиву* командой ReDim. Причем, для повторного использования *динамического массива*, придется снова устанавливать его размерности. Если команда Erase применяется к объектному *массиву*, в каждый его элемент записывается специальное значение Nothing, которое означает пустую ссылку на объект.

Теперь, когда мы обсудили циклы For-Next и работу с *массивами*, поговорим о других типах циклов.

**7.9. Цикл с предусловием**

[07-07-Цикл с предусловием.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/7/files/07_07.docm)- пример к п. 7.9.

Как вы уже знаете, *цикл с предусловием* While - Wend выполняется до тех пор, пока условие, указанное на входе, верно.

В листинге 7.11. представлено решение такой задачи: выводить на экран случайные числа от 0 до 20 до тех пор, пока не будет выведено число больше 10.

А = 1

While А < 10

А = Int(Rnd() \* 20)

MsgBox А

Wend

Листинг 7.11. Вывод случайных чисел в цикле While - Wend

Сначала мы приравниваем переменной А число 1. Проверка при входе в цикл находит, что А меньше 10 и запускает первый проход. Переменной А приравнивается целое случайное число, это число выводится в окне сообщения. Дальше следует новая проверка - если A все еще меньше 10 - все повторяется снова. Если A больше или равно 10 - число не выводится, после чего выполнение цикла прекращается.

Теперь рассмотрим *цикл с постусловием*.

**7.10. Цикл с постусловием**

[07-08-Цикл с постусловием.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/7/files/07_08.docm)- пример к п. 7.10.

Цикл Do-Loop While выполняется до тех пор, пока значение на выходе из цикла не верно. Подобные циклы используют, например, для проверки правильности ввода каких-либо данных пользователем. Если данные введены неверно - цикл выполняется снова.

Аналогично действует цикл Do-Loop Until - он будет выполняться до тех пор, пока условие цикла неверно (то есть равно False ).

В листинге 7.12. вы можете найти пример такого цикла. Здесь пользователю предлагается ввести какое-нибудь число. Если введено не число (то есть функция IsNumeric возвратит False ), программа выведет окно ввода снова.

Dim var\_A

Do

var\_A = InputBox("Введите число")

Loop Until IsNumeric(var\_A)

Листинг 7.12. Проверка ввода в цикле с постусловием

**7.11. Принятие решений: If-Then-Else**

[07-09-If-Then-Else.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/7/files/07_09.docm)- пример к п. 7.11.

Программы на VBA умеют принимать решения - для этого существуют операторы условного перехода. Они объединены в конструкцию If - Then - Else.

В этой конструкции могут быть использованы следующие операторы сравнения ([табл. 7.4](https://www.intuit.ru/studies/courses/494/350/lecture/8363?page=2#table.7.4).)

|  |  |
| --- | --- |
| Таблица 7.4. Операторы сравнения | |
| **Оператор** | **Описание** |
| = | Равно |
| <> | Не равно |
| > | Больше |
| < | Меньше |
| >= | Больше или равно |
| <= | Меньше или равно |
| Like | Сравнение строки с шаблоном |
| Is | Сравнение объектов |

Напишем простую программу ([листинг 7.13](https://www.intuit.ru/studies/courses/494/350/lecture/8363?page=2#example.7.13).), которая спрашивает у пользователя его возраст. Если введенный возраст меньше 18 - программа должна вывести надпись "Вам менее 18 лет", если больше или равен 18 - надпись "Вам 18 или больше".

a = InputBox("Введите ваш возраст")

If a < 18 Then MsgBox ("Вам меньше 18")

If a >= 18 Then MsgBox ("Вам 18 или больше")

Листинг 7.13. Оператор If - Then

Здесь представлен оператор в простейшем виде - проверка условия и выполнение однострочной команды. А что если нужно выполнить не одну команду, а несколько? Для этого служит команда End If ([листинг 7.14](https://www.intuit.ru/studies/courses/494/350/lecture/8363?page=2#example.7.14).)

a = InputBox("Введите ваш возраст")

If a < 18 Then

MsgBox ("Вам меньше 18")

MsgBox ("Вам не следует смотреть этот фильм")

End If

If a >= 18 Then MsgBox ("Добро пожаловать")

Листинг 7.14. Использование команды End If

Здесь программа выводит два сообщения, если пользователю меньше 18 лет.

Оператор может иметь вид If - Then - Else ([листинг 7.15](https://www.intuit.ru/studies/courses/494/350/lecture/8363?page=2#example.7.15)). Благодаря ему два оператора If - Then из листинга 7.13. можно объединить в один.

a = InputBox("Введите ваш возраст")

If a < 18 Then MsgBox ("Вам меньше 18") \_

Else MsgBox ("Вам больше 18")

Листинг 7.15. Использование команды Else

Применение команды Else так же позволяет исполнять многострочные команды. Например, вот так - [листинг 7.16](https://www.intuit.ru/studies/courses/494/350/lecture/8363?page=2#example.7.16).

a = InputBox("Введите ваш возраст")

If a < 18 Then

MsgBox ("Вам меньше 18")

MsgBox ("Вам не следует смотреть этот фильм")

Else

MsgBox ("Вам больше 18")

MsgBox ("Добро пожаловать")

End If

Листинг 7.16. Выполнение многострочных команд

В операторе If возможно использование сложных условий. Например, вы просите пользователя ввести имя и пароль. Если они соответствуют данным, хранящимся в системе, программа выводит приветствие, иначе - сообщает о том, что пользователь ввел неправильные сведения. Очевидно, что нам нужно проверить два условия, причем важно, чтобы и то и другое выполнялось. Для этого можно воспользоваться *логическим оператором* And, который трактуется как "И". Конструкция с And выглядит так - [листинг 7.17](https://www.intuit.ru/studies/courses/494/350/lecture/8363?page=2#example.7.17).

UserName = InputBox("Введите имя пользователя")

UserPass = InputBox("Введите ваш пароль")

If UserName = "Александр" And UserPass = "12345" Then

MsgBox ("Добро пожаловать в систему")

Else

MsgBox ("Неверное имя пользователя или пароль")

End If

Листинг 7.17. Использование And

Словесно вышеприведенную конструкцию можно описать так: " Если имя равно Александр и пароль равен 12345…".

Логический оператор Or (переводится как "Или") позволяет принять положительное решение, если выполняется хотя бы одно из условий. Например, нам нужно узнать имя пользователя, и если оно соответствует одному из имен, зарегистрированных в системе, вывести приветствие, иначе - вывести информацию об отсутствии в системе такого пользователя - листинг. 7.18.

UserName = InputBox("Введите имя пользователя")

If UserName = "Александр" Or UserName = "Сергей" Or \_

UserName = "Николай" Then

MsgBox ("Добро пожаловать в систему, " & UserName)

Else

MsgBox ("В системе нет такого пользователя!")

End If

Листинг 7.18. Использование Or

And и Or можно использовать вместе. Оператор Not (Не) позволяет задавать условия с отрицанием. Например, вы можете пропустить в систему всех пользователей кроме пользователя с именем "Владимир".

Для проверки дополнительных условий можно использовать оператор If - Then - Else.

Выше, в [табл. 7.4](https://www.intuit.ru/studies/courses/494/350/lecture/8363?page=2#table.7.4)., есть пара необычных операторов - Like для сравнения строк с шаблоном, и Is - для сравнения *объектных переменных*.

**7.12. Сравнение с использованием Like и Is**

[07-10-Сравнение строк с шаблонами.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/7/files/07_10.docm)- пример к п. 7.12.

Оператор Like используется для сравнения строк с шаблонами. Шаблон - это особым образом записанная последовательность символов. При построении шаблонов используются специальные символы, приведенные в [табл. 7.5](https://www.intuit.ru/studies/courses/494/350/lecture/8363?page=3#table.7.5).

|  |  |
| --- | --- |
| Таблица 7.5. Символы для построения шаблонов | |
| **Символы** | **Описание** |
| ? | Любой одиночный символ |
| \* | Любое количество любых символов |
| # | Любая одиночная цифра |
| [список символов] | Любой одиночный символ, входящий в список символов |
| [!список символов] | Любой одиночный символ, не входящий в список |

После построения шаблона его заключают в кавычки.

Давайте рассмотрим пример, реализующий следующие проверки.

* Узнать, есть ли в строке прописные и заглавные буквы латинского алфавита.
* Проверить, состоит ли введенное слово из четырех символов (цифр или букв)
* Проверить, состоит ли введенная последовательность из двух любых символов (цифр или букв) и двух цифр
* Проверить, нет ли во введенной строке русских букв " а " и " о "
* Если введенная строка начинается двумя буквами " d " и заканчивается тремя буквами " f ", сообщить об этом

В листинге 7.19. вы можете найти решение этой задачи

Dim str\_Inp As String

str\_Inp = InputBox("Введите строку")

'Есть ли латинские буквы в строке

If str\_Inp Like "\*[a-z]\*" Or \_

str\_Inp Like "\*[A-Z]\*" \_

Then MsgBox ("В строке есть латинские буквы")

'Состоит ли введенное слово из 4-х символов

If str\_Inp Like "????" Then \_

MsgBox ("Введенное слово состоит из 4-х символов")

'Состоит ли введенная последовательность

'из 2-х любых символов и 2-х цифр

If str\_Inp Like "??##" Then \_

MsgBox ("Введены два любых символа и две цифры")

'Проверка на отсутствие букв

'"а", "о"

If str\_Inp Like "[!а]" And \_

str\_Inp Like "[!о]" Then

MsgBox ("В строке нет букв " + Chr(34) + \_

"а" + Chr(34) + " и " + Chr(34) + "о" + Chr(34))

End If

'Проверка на наличие в начале

'введенной строки двух букв "d", а в конце

'трех "f"

If str\_Inp Like "dd\*fff" \_

Then MsgBox ("Строка имеет вид: dd\*fff")

Листинг 7.19. Использование Like

Как вы можете видеть, шаблон для сравнения с текстом всегда заключается в двойные кавычки. На основе информации об использовании шаблонных символов вы можете самостоятельно построить выражения для проверки различных последовательностей.

Оператор Is используется для работы с *объектными переменными*. Мы будем подробно рассматривать их в начале следующей главы. Оператор Is проверяет, ссылаются ли две объектные переменные на один и тот же объект. Если это так - он возвращает True, если нет - то False.

**7.13. Принятие решений: Select Case**

[07-11-Select Case.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/7/files/07_11.docm)- пример к п. 7.13.

Если вам предстоит проверить данные на множество значений, для каждого из которых надо выполнить какое-то особое действие, можно использовать множество операторов If или один Select Case.

В листинге 7.20. приведен пример использования Select Case - в зависимости от введенного имени программа здоровается с пользователями по-разному.

Dim str\_UserName As String

str\_UserName = InputBox("Введите имя пользователя")

Select Case str\_UserName

Case "Александр"

MsgBox ("Привет")

Case "Сергей"

MsgBox ("Здравствуй")

Case "Николай"

MsgBox ("Добро пожаловать")

Case Else

MsgBox ("Не существует такого пользователя")

End Select

Листинг 7.20. Использование Select Case

После Select Case указано имя переменной, анализ которой осуществляется. Дальше расположено произвольное количество вариантов - каждый вариант после ключевого слова Case. Если не выполнен ни один из Case - выполняется Case Else. В конце этой конструкции находится оператор End Select.

**7.14. Оператор безусловного перехода**

[07-12-GoTo.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/7/files/07_12.docm)- пример к п. 7.14.

В VBA есть один оператор, которым не рекомендуется пользоваться при написании программ. Это *оператор безусловного перехода* GoTo. Он позволяет передать управление в определенное место программы, заданное номером строки или меткой. Меткой строки может быть любое слово (подчиняющееся правилам *именования переменных*), после которого следует двоеточие.

Например, *цикл с постусловием* можно создать таким способом ([листинг 7.21](https://www.intuit.ru/studies/courses/494/350/lecture/8363?page=3#example.7.21).):

Dim num\_Cycle

num\_Cycle = 0

Cycle\_Start:

num\_Cycle = num\_Cycle + 1

MsgBox ("Проход цикла номер: " + Str(num\_Cycle))

If num\_Cycle < 5 Then GoTo Cycle\_Start

Листинг 7.21. Цикл с постусловием с помощью оператора GoTo

Здесь мы создаем метку Cycle\_Start, которая означает начало цикла. После этого увеличиваем на единицу переменную, которая является счетчиком *цикла, выводим* ее в окне сообщения. В последней строке мы проверяем переменную. Если она меньше 5 - команда GoTo Cycle\_Start передает управление на соответствующую метку. В такой простой конструкции, как наша, особенных сложностей в чтении кода не видно. Но стоит программе хотя бы немного увеличиться, несколько подобных циклов (особенно - вложенных, а еще хуже - использующих номер строки для перехода) превратят ее в сложный для чтения и правки текст.

GoTo делает программы трудночитаемыми, его практически всегда можно заменить, используя другие программные конструкции. Единственное место, где GoTo пока незаменим - это использование его при написании обработчиков ошибок времени выполнения. В следующей лекции мы поговорим о таких ошибках и об использовании GoTo для организации подпроцедур в исполняемой процедуре.

**7.15. Работа с файлами**

[07-13-Работа с файлами.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/7/files/07_13.docm)- пример к п. 7.15.

Программы, написанные на VBA, умеют работать с внешними *файлами*. В частности, на практике могут возникнуть задачи по поиску *файлов* в директориях, по открытию, обработке, сохранению *файлов*. Открытие, обработка, сохранение - дело отдельных приложений (например, MS Word, MS Excel) - то есть эти задачи решаются с помощью объектных моделей этих приложений. А вот поиск *файлов* осуществляется общими для всех методами VBA.

Как правило, чтобы открыть *файл*, нужно знать его имя. Иными словами, поиск *файлов* заключается в получении имен *файлов*, находящихся в определенной директории. Для этого можно использовать команду Dir. Она возвращает строку, содержащую имя *файла*, используя путь, заданный при вызове. Давайте рассмотрим конструкцию ([листинг 7.23](https://www.intuit.ru/studies/courses/494/350/lecture/8363?page=3#example.7.23).), которая позволяет найти все *файлы*, находящиеся в корневой директории диска C.

var\_Doc = Dir("C:\\*.\*")

Do While var\_Doc <> ""

MsgBox var\_Doc

var\_Doc = Dir()

Loop

Листинг 7.23. Поиск всех файлов в корневом каталоге диска C

Сначала мы присваиваем переменной var\_Doc первое найденное имя *файла*. Очевидно, что узнав имя *файла*, мы можем сказать, что нашли этот файл на диске. В самом простом варианте использования функции Dir в качестве параметров мы передаем ей путь и маску имени *файла*. Знак \* в маске означает любое количество любых символов. Следовательно, \*.\* означает "все файлы" - то есть *файлы* с любыми именами и любыми расширениями. В маске можно так же использовать знак ? - он символизирует один любой символ. Если не указать путь к *файлам*, а лишь маску - Dir будет искать их в *текущей директории*. Например, для Microsoft Word по умолчанию это папка Мои документы.

Помимо пути и маски при поиске *файлов* можно указать некоторые дополнительные параметры. Так, по умолчанию функция ищет лишь обычные *файлы*, не обращая внимания на папки, скрытые и системные *файлы*. Чтобы функция нашла по заданному пути не только *файлы*, но и папки, ее нужно вызвать так:

var\_Doc = Dir("C:\\*.\*", vbDirectory)

Обратите внимание на то, что после пути и маски указан параметр vbDirectory - он указывает функции, что она должна включить в поиск и директории.

После того, как первое найденное имя присвоено переменной, мы запускаем *цикл с предусловием*, в котором проверяем, не пуста ли эта переменная. Если Dir не обнаружил по указанному пути ничего подходящего под заданную маску, он возвратит, пустую строку. Следовательно, цикл в таком случае не выполнится ни разу.

В цикле есть две строки. Первая выводит найденное имя на экран, а вторая - вызывает функцию Dir еще раз - без параметров. Такой вызов возвращает следующее имя *файла*, подходящее под заданный при первом вызове Dir шаблон. После этого все повторяется. В реальной программе в такой цикл можно вставить команды для работы с найденными *файлами*.

Помимо Dir полезной может оказаться команда ChDir. Она позволяет перейти в указанную при ее вызове директорию, которая будет использоваться в качестве директории по умолчанию. Такая конструкция, предшествующая циклу из предыдущего примера позволит найти все *файлы* в папке "Документы", которая расположена по пути " C:\Документы ":

ChDir ("C:\Документы")

var\_doc = Dir("\*.\*")

Существует множество других функций работы с *файлами*, вы можете найти информацию о них в справочной системе VBA.

**7.16. Выводы**

В этой лекции мы познакомились с *массивами*, циклами, *операторами принятия решений*, обсудили основные положения, касающиеся работы с *файлами*. Следующая лекция посвящена отладке программ и обработке ошибок.

Лекция 8. Отладка приложений, обработка ошибок

**8.1. Ошибки при создании программы**

Можно выделить два типа ошибок, с которыми сталкивается программист. Во-первых - это ошибки, которые сопровождают создание программ, а во вторых - *ошибки времени выполнения*.

Если вы неправильно введете оператор или *ключевое слово*, если забудете указать часть выражения - ошибка сразу же будет заметна в редакторе. Такие ошибки относятся к *синтаксическим*. Также на этапе разработки можно обнаружить и устранить *логические ошибки*. Они связаны с неправильными формулами расчета показателей, неверным использованием переменных и т.д.

Приведем несколько примеров *синтаксических ошибок*.

* Неправильное ключевое слово. Такая ошибка сразу будет отмечена редактором. Например, если в конструкции цикла For-Next сделать ошибку в ключевом слове For - вы увидите сообщение об ошибке как только попытаетесь перейти к написанию следующей, после начала цикла, строки.
* Ошибка при объявлении переменной. Если вы забудете указать ключевое слово Dim при объявлении переменной - редактор выдаст сообщение об ошибке. Однако, сделает он это лишь при попытке запуска программы.

А вот - *логические ошибки*.

* Неправильное использование операторов. Например, вместо знака обычного деления вы случайно использовали знак целочисленного деления.
* Расчет какого-либо показателя по неправильной формуле. Например, если неточно расставить скобки в каком-либо выражении - это приведет к ошибке, хотя внешне все может выглядеть правильным.
* Неправильное использование функций - как встроенных, так и пользовательских. Например, используя функцию Str для получения строкового представления числа, вы не учли, что для положительных чисел эта функция добавляет в начало строки пробел. Далее вы попытались узнать первую цифру числа, вырезав первый символ полученной строки. Естественно, никакой цифры в этом случае не получится - лишь знак пробела для положительных или "минус" для отрицательных чисел.
* Неправильное использование переменных. Например, вы используете два *вложенных цикла* для обработки двумерного массива. Одна из цикловых переменных имеет имя i, вторая - j. Они довольно сильно похожи внешне, их можно случайно перепутать при указании индексов массива. К тому же, обрабатывая массив в цикле довольно легко перепутать место каждой из переменных при указании *индекса массива*. Использование понятных имен переменных (например - my\_Age или num\_Vozrast для хранения возраста и т.д.) позволяет эффективно бороться с такими ошибками.
* Случайное использование "новых" переменных. Например, вы предложили пользователю ввести некое значение и записали его в переменную num\_Inp, а использовав эту переменную в выражении, напечатали не num\_Inp, а num\_Ihp. Внешне они похожи, но, присмотревшись, вы можете обнаружить, что имена разные. Еще сложней искать ошибки в латинских именах переменных, в которые "вкрались" русские буквы. Разницу между my\_Name и my\_Namе вы не увидите, но это - разные переменные - в конце второй вместо латинской e использована русская е. Эффективно бороться с такими ошибками можно, если задать в редакторе опцию обязательного объявления переменных ( Option *Explicit* ). При появлении необъявленной переменной редактор даст знать об этом.
* Неправильное использование оператора сравнения. Например, это может быть оператор, который сравнивает некие величины не так, как вы предполагали - вместо знака < вы случайно использовали > или, редактируя сравнение (скажем, скопировав похожий оператор сравнения из другого места программы для ускорения работы), поменяли местами сравниваемые переменные или выражения, не поменяв знака и т.д.

Ошибки логики можно найти лишь тщательно проверив и протестировав программу. Чтобы облегчить работу с кодом программы, рекомендуется снабжать ее комментариями. Вовсе необязательно комментировать каждую строчку, достаточно выделить крупные функциональные блоки.

**8.2. Ошибки при выполнении программы**

Такие ошибки называют еще *ошибками времени выполнения*. Происходят они, как правило, при неправильном вводе данных пользователем, при возникновении обстоятельств, делающих дальнейшую нормальную работу программы невозможной. Например, ошибку вызовет попытка использовать текстовые данные в арифметическом выражении, попытка сохранения файла в несуществующей директории, *деление* на ноль и т.д.

*Ошибки времени выполнения возникают* в нормально работающих программах, которые прошли проверку на синтаксическую и логическую правильность. С этими ошибками можно бороться используя один из двух методов. Первый - разработка программы таким образом, чтобы не допустить этих ошибок, создание программных конструкций, которые предотвращают возникновение ошибок. Второе - перехват ошибок и их обработка.

Давайте, для начала, рассмотрим способы поиска ошибок в процессе создания программ

**8.3. Тестирование программ и поиск ошибок**

*Синтаксические ошибки*, как уже было сказано, выявляются обычно на очень раннем этапе - сразу после набора текста в редакторе. А вот для поиска *логических ошибок* полезно протестировать программу.

Для начала протестируйте вашу программу в обычном режиме работы. Например, если вы предлагаете ввести пользователю число - введите сами число и проверьте результаты работы. Если действия программы будут различаться в зависимости от введенных данных - попытайтесь перебрать все варианты этих данных - так, чтобы проверить все ветви программы.

Если обычные проверки закончились успешно - можно считать, что ваша *программа* корректно работает, не имеет *синтаксических* и *логических ошибок*.

После этого можно усложнить режим тестирования, чтобы как можно более полно сымитировать действия пользователя программы и посмотреть на ее поведение. Еще лучше - попытайтесь целенаправленно вызвать ошибку. Гораздо лучше, когда *программа* даст сбой на вашем рабочем столе, и вы сможете оперативно устранить этот сбой, чем ждать, когда то же самое произойдет у другого пользователя.

Для проверки программы в реальных условиях попробуйте провести следующие эксперименты.

* Если программа запрашивает число - введите какое-нибудь слово, очень большое число, ноль, отрицательное число, оставьте поле ввода пустым, введите дробное число. Если вы не предусмотрели никаких специальных мер по обработке ошибок, при выполнении подобных операций вы почти гарантированно встретитесь с проблемами.
* Попытайтесь запустить программу, открыв несколько окон с документами.
* Попытайтесь прервать работу программы, а потом снова возобновить ее. Если в вашей программе есть участки, в течение выполнения которых нельзя допускать прерывания работы программы пользователем - вам следует подумать о том, чтобы запретить прерывание работы программы на этих участках.
* Попытайтесь использовать вашу программу в более старой версии Microsoft Office, в нерусифицированной версии. Обратите внимание, например, на различия расширений имен файлов в Office 2007 и более старых версиях, на различия в объектных моделях. В общем случае программы из более старых версий Office будут работать в Office 2007, однако если они используют какие-то специфические особенности Office - такие программы нуждаются в проверке и обновлении. В то же время, макросы для Office 2007, использующие новые объектные модели, могут не работать или работать неправильно в старых версиях.
* Попытайтесь поработать с вашей программой на чужом ПК. Вполне возможно, что при таком эксперименте вы столкнетесь с ошибкой. Например, вы программно работаете с файлами на вашем рабочем ПК - если эти файлы отсутствуют на ПК другого пользователя или находятся в других директориях, или тех директорий, которые нужны вашей программе, нет на ПК другого пользователя - вы столкнетесь с ошибкой.
* Во время работы программы сделайте что-нибудь необычное. Как правило, от пользователей можно ожидать любых странных на первый взгляд действий. Если вы тестируете программу для MS Word, которая правит текст или занимается автоматическим созданием текста, попробуйте во время ее работы переключаться между документами, читать документ, вносить в него правки, выделять произвольные участки текста. То же самое касается MS Excel - во время работы программы попробуйте переключаться между открытыми книгами, между листами, выделять ячейки, попробуйте запустить программу, делая различные листы активными, открыв несколько книг. Результаты такого тестирования могут быть совершенно непредсказуемыми. Проанализировав их, вы можете прити к выводу, что, например, на время выполнения программы нужно скрывать или блокировать документ, пользоваться альтернативными методами работы с документом. Яркий пример - объект Selection в MS Word, который чувствителен к смене выделения в процессе работы, и объект Range, который может работать совсем без создания выделения в тексте.
* Если ваша программа использует файлы, находящиеся в локальной сети, отключите сеть во время работы программы. Проверьте ее реакцию. То же самое можно сделать, если ваша программа работает с принтером - проверьте ее реакцию на выключенный принтер, на принтер, в котором нет бумаги.
* Наконец, представьте, что вы - пользователь программы и просто поработайте с ней. А еще лучше - попросите потенциального пользователя немного "пообщаться" с вашей программой. Этот способ позволяет протестировать программу в условиях, максимально приближенных к реальным.

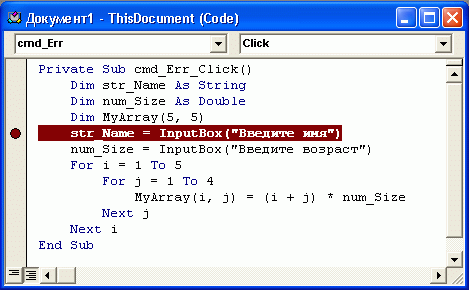
Возможно, вам покажется, что такая проверка не нужна вашей программе. Если вы пишете небольшой *макрос* для собственного использования, скорее всего, так оно и есть. Но стоит вашему проекту хоть немного вырасти, методы поиска и устранения ошибок могут оказаться очень кстати.

Если вы встретитесь с ошибками и затрудняетесь определить, где именно они происходят - попробуйте выполнить программу в пошаговом режиме, используйте встроенные в редактор *VBA* средства для *отладки программ*.

**8.4. Отладка программ в редакторе VBA**

Основной метод отладки - это *пошаговое исполнение программы* с использованием точек останова (*breakpoint*).

Чтобы создать в программе точку останова, достаточно щелкнуть мышью в редакторе на серой панели напротив команды, на которой нужно остановить выполнение программы. Там появится большая красная точка ([рис. 8.1](https://www.intuit.ru/studies/courses/494/350/lecture/8364?page=1#image.8.1).) - здесь *программа* будет остановлена в процессе выполнения. Строка будет подсвечена красным цветом.



**Рис. 8.1.**Точка останова в программе

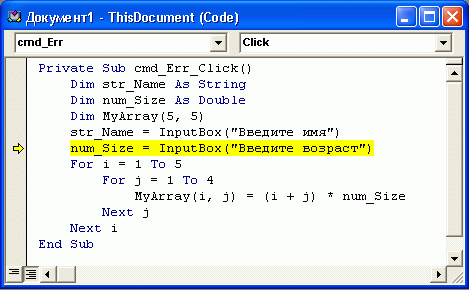
Следует понимать, что строка, подсвеченная при остановке программы еще не выполнялась - редактор указывает на нее, как бы говоря "Эта строка будет выполнена следующей".

Таких точек останова можно установить столько, сколько нужно - на тех строках программы, где вы подозреваете возникновение ошибки. Чтобы убрать точку останова, щелкните по ней мышью.

Установить точку останова в строку можно, выделив строку и выбрав команду **Debug o Toggle Breakpoint**(*Отладка* o Установить точку останова). Для удаления всех точек останова из программы можно воспользоваться командой **Debug o Clear All Breakpoints** (*Отладка* o Очистить точки останова).

Вместо точек останова, расставленных мышью или из *меню*, можно использовать оператор Stop. Он останавливает работу программы и переводит ее в режим *отладки*.

Так же режим *отладки* можно включить, нажав во время работы программы комбинацию клавиш *Ctr* + Pause Break и нажав в появившемся окне кнопку Debug. Текущая строка будет выделена желтым цветом, напротив нее будет установлена желтая стрелочка. Однако такой способ обычно не позволяет точно "попасть" в то *место* программы, где находится предполагаемая ошибка ([рис. 8.2](https://www.intuit.ru/studies/courses/494/350/lecture/8364?page=1#image.8.2).).

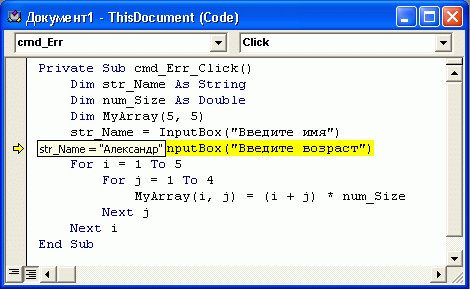


**Рис. 8.2.**Программа, остановленная во время выполнения

Чтобы запущенная *программа* останавливалась на каждой строке, можно запустить ее в режиме Step Into командой Debug o Step Into (*Отладка* o Пошаговое *исполнение*). Того же эффекта можно достичь, нажав клавишу F8 на клавиатуре.

Когда *программа* остановлена, вы можете выполнить следующие действия

* Просмотреть значения переменных, наведя на них указатель мыши. Например, на [рис. 8.3](https://www.intuit.ru/studies/courses/494/350/lecture/8364?page=1#image.8.3). вы видите всплывающее окно, которое содержит значение переменной.



**Рис. 8.3.**Значение переменной во всплывающем окне

* Продолжить выполнение программы в режиме **Step Into** - выбрав соответствующую команду меню или нажав клавишу **F8**.
* Отредактировать программу.
* Продолжить исполнение программы в обычном режиме командой **Run o Sub/User Form** (Запустить o Процедуру/Форму), нажатием клавиши **F5** или соответствующей кнопкой на панели инструментов
* Остановить выполнение программы командой **Run o Reset** (Запустить o Перезагрузка) или кнопкой на панели инструментов
* Воспользоваться другими средствами *отладки* - окнами **Immediate**, **Locals**, **Watch**.

Помимо режима **Step Into** существуют следующие режимы отладки, доступные в *меню* **Debug**.

* **Step Over** (Перейти на следующую строку). Эта команда полезна при *отладке программы*, содержащей вызовы уже отлаженных процедур. В режиме **Step Over** отладчик не *входит в процедуру*, выполняя ее без *отладки*, после чего переходит на следующую строку. Например, вы выполняете программу в режиме **Step Into** и при очередной остановке видите, что подсвеченная строка содержит вызов процедуры, которую отлаживать не нужно. Вы выбираете команду **Step Over**, процедура выполняется без остановок на каждой ее строке, после чего следующая остановка происходит на строке вашей программы, которая идет за вызовом процедуры.
* **Step Out** (Выполнить процедуру) - эта команда позволяет выполнить текущую процедуру (например, вызванную из кода основной программы при обычной *отладке* ) без остановки в каждой строке. Следующая остановка будет сделана на строке, которая следует за вызовом процедуры в основном тексте программы.
* **Run To Cursor** (Выполнить до курсора) - выполняет программу до позиции, на которой установлен курсор. Аналогично установке одиночной точки останова.

Помимо точек останова существуют и другие средства *отладки*. Они полезны при проверке значений переменных (ведь если переменных достаточно много - проверка их значений в коде программы может превратиться в утомительное и непродуктивное занятие), свойств объектов, которые могут вызвать ошибки и в других случаях.

**8.5. Immediate**

[08-01-Работа с Immediate.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/8/files/08_01.docm) - пример к п. 8.5.

Окно **Immediate** (Немедленное выполнение) представляет собой нечто вроде "филиала" редактора кода. Чтобы отобразить его, можно воспользоваться командой **View o Immediate Window** (Вид o Окно непосредственных изменений). С помощью **Immediate** можно мгновенно выполнять различные команды, вызывать процедуры и функции - как пользовательские, так и встроенные. Так же в него можно организовать *вывод* различных данных в ходе выполнения программы. Давайте рассмотрим основные возможности этого окна.

Для отправки в окно **Immediate** информации из выполняющейся программы нужно воспользоваться такой конструкцией:

Debug.Print

Метод Print объекта Debug организует *вывод* в **Immediate**. После того, как выполнение программы завершится, вы сможете проанализировать выведенные данные.

Например, добавим в *VBA*-проект форму, назовем ее frm\_Test, на форму добавим кнопку - cmd\_Test. Добавим в код обработчика нажатия кнопки такой код ([листинг 8.1](https://www.intuit.ru/studies/courses/494/350/lecture/8364?page=2#example.8.1).):

Dim TwoDim(4, 5)

Dim OneDim(30)

Dim num\_Transfer

For i = 0 To 4

For j = 0 To 5

TwoDim(i, j) = Int(Rnd \* 100)

Next j

Next i

For i = 0 To 4

Debug.Print ("i=" + Str(i))

For j = 0 To 5

num\_Transfer = num\_Transfer + 1

OneDim(num\_Transfer) = TwoDim(i, j)

Debug.Print ("j=" + Str(j))

Debug.Print ("OneDim" + Str(num\_Transfer) + \_

" =" + Str(OneDim(num\_Transfer)))

Debug.Print ("TwoDim" + Str(i) + Str(j) + \_

" =" + Str(TwoDim(i, j)))

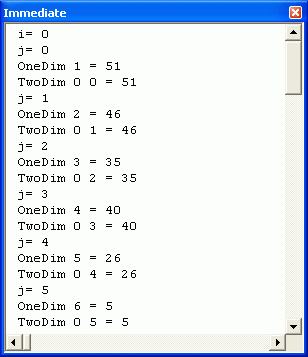
Next j

Next i

Листинг 8.1. Обработчик события Click кнопки cmd\_Test

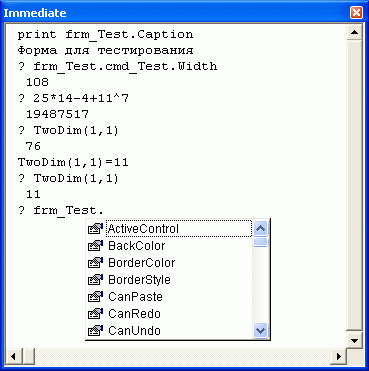
Здесь мы объявили два массива - один двумерный ( TwoDim ), второй одномерный ( OneDim ). Заполнили TwoDim случайными числами, после чего организовали перенос данных из него в OneDim. В циклическую конструкцию, которая осуществляет перенос данных, мы вставили несколько вызовов Debug.Print. Они позволили нам получить полную картину работы этого блока в окне **Immediate** ([рис. 8.4](https://www.intuit.ru/studies/courses/494/350/lecture/8364?page=2#image.8.4).).

Подобный же метод может быть весьма полезен, например, при проверке алгоритма сортировки и любых других сложных алгоритмов.



**Рис. 8.4.**Окно Immediate после вывода значений

На [рис. 8.5](https://www.intuit.ru/studies/courses/494/350/lecture/8364?page=2#image.8.5). вы можете видеть окно **Immediate**, которое содержит несколько команд. Эти команды мы вводили в режиме останова программы из листинга 8.1.



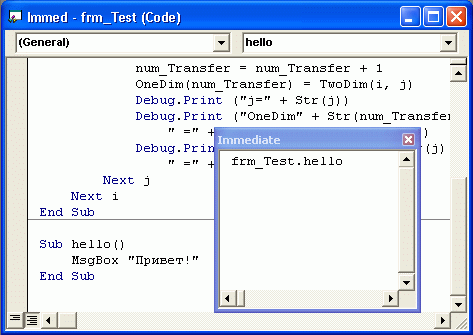
**Рис. 8.5.**Работа с Immediate в режиме останова

*Команда* Print frm\_Test.Caption вывела свойство Caption формы frm\_Test в окно. Как видите, **Immediate** предлагает справку по объектам точно так же, как и обычный редактор. Далее, обратите внимание на то, что вместо команды Print можно использовать знак ?. Если вы программировали на обычном Basic'e, этот значок должен быть вам хорошо знаком.

Командой ? 25\*14-4+11^7 мы вычисляем введенное *выражение*.

Командой ? TwoDim(1, 1) мы выводим число, хранящееся в TwoDim(1, 1). В нашем случае это 76. После этого мы записываем команду TwoDim(1, 1)=11, тем самым присваивая элементу массива новое *значение*, и выводим это *значение* снова.

Теперь рассмотрим возможности по запуску процедур и функций. Создадим в коде формы простую процедуру с именем hello, которая выводит окно сообщения с текстом "Привет". Чтобы вызывать эту процедуру, не выполняя программу, достаточно ввести в окно **Immediate** такой код: frm\_Test.hello. Вот, как выглядит окно после ввода команды ([рис. 8.6](https://www.intuit.ru/studies/courses/494/350/lecture/8364?page=2#image.8.6).).



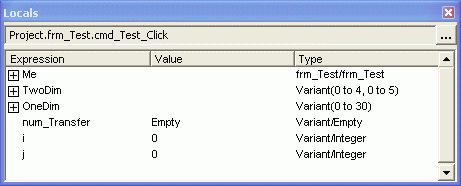
**Рис. 8.6.**Окно кода с пользовательской процедурой и Immediate после ввода команды

После нажатия на кнопку **Enter** процедура будет выполнена и окно сообщения покажется на экране. Таким способом можно проверять работу процедур, отлаживать их до включения их в основной *код программы*.

Как видите, **Immediate** - это очень простое, но мощное средство *отладки*. Однако, его неудобно использовать для быстрого просмотра *множества* свойств элементов управления, значений многих переменных и т.д. В подобных случаях нам может помочь окно **Locals**.

**8.6. Locals**

Окно **Locals**(Локальные переменные) позволяет просматривать и изменять свойства всех объектов приложения и значения переменных. Посмотрите, как выглядит окно **Locals** при открытии его в режиме останова программы, с которой мы работали в п. 8.5. ([рис. 8.7](https://www.intuit.ru/studies/courses/494/350/lecture/8364?page=2#image.8.7).)



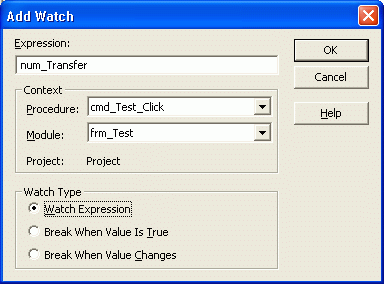
**Рис. 8.7.**Окно Locals открытое при останове программы

Как видите, здесь есть *объект* Me, который представляет собой форму, массивы, объявленные в коде, переменные. Для посмотра или редактирования свойств объектов и содержимого массивов, достаточно развернуть их описания, нажав на знак + перед их наименованиями. Для редактирования значения нужно просто выделить его и ввести вместо старого новое. Учтите, что строковые значения нужно вводить в кавычках - так же, как и в программном коде.

**8.7. Watches**

Окно Watches (Контролируемые выражения) предназначено для контроля за отдельными выражениями, значениями переменных и т.д.

Чтобы добавить такое *выражение*, щелкните мышью по нему и выберите команду *меню* **Add Watch** (Контролировать). Появится окно **Add Watch** ([рис. 8.8](https://www.intuit.ru/studies/courses/494/350/lecture/8364?page=2#image.8.8).).



**Рис. 8.8.**Окно Add Watch

Здесь, в *поле* **Exression** (*Выражение*) нужно ввести контролируемое *выражение*. В нашем случае это *имя переменной*.

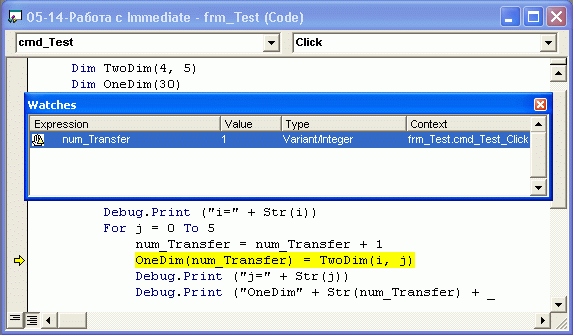
В группе параметров **Context** (*Контекст*) следует выбрать область контроля. Чем эта область меньше (в нашем случае - это процедура cmd\_Test\_Click, которая хранится в модуле формы frm\_Test ) - тем лучше.

В группе параметров Watch type (Тип наблюдения) нужно выбрать тип наблюдения за выражением.

**Watch Expression** (Наблюдать за выражением) - *программа* просто выводит в окно **Watch** *значение* наблюдаемого выражения, ничего не предпринимая. Если вы перейдете в режим *отладки*, то сможете просмотреть это *выражение*.

**Break When Value Is True** (Прервать работу программы если *выражение* выполняется) - при выполнении определенного в окне **Add Watch** условия *программа* остановится и перейдет в режим *отладки*. Например, если мы напишем в *поле* **Expression** *выражение* num\_Transfer > 100 и активируем данный *пункт*, то работа программы будет остановлена лишь тогда, когда *значение переменной* num\_Transfer превысит 100.

**Break When Value Changes** (Прервать работу программы при изменении значения) - прерывает выполнение программы если *значение* контролируемой переменной или выражения изменилось. Если мы сделаем эту опцию активной, при каждом изменении переменной *программа* будет переходить в режим *отладки*. Установим эту опцию и запустим программу. При первом же изменении переменной num\_Transfer мы попадем в режим *отладки* ([рис. 8.9](https://www.intuit.ru/studies/courses/494/350/lecture/8364?page=2#image.8.9).).



**Рис. 8.9.**Работа с окном Watches в режиме отладки

Как видите, в окне **Watches** отображены следующие данные:

* **Expression** - контролируемое выражение.
* **Value** - значение выражения.
* **Type** - тип данных.
* **Context** - контекст (область контроля).

При необходимости вы можете отредактировать контролируемое *выражение* - для этого щелкните правой кнопкой мыши по строке выражения в окне **Watches** и выберите *пункт* **Edit Watch**. Для удаления выражения выберите в том же *меню* *пункт* **Delete Watch**, для добавления нового - **Add Watch**.

**8.8. Проверка данных, вводимых пользователем**

[08-02-Проверка введенных даных.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/8/files/08_02.docm) - пример к п. 8.8.

Для проверки данных, вводимых пользователем, вы можете воспользоваться стандартными конструкциями для *принятия решений*.

Давайте рассмотрим пример, в котором *пользователь* должен ввести свой возраст в ответ на *запрос* программы. Очевидно, что возраст должен удовлетворять следующим условиям.

* должен быть числом
* ограничим введенное значение диапазоном значений для типа данных Byte - то есть введенное значение не может быть больше 256 и меньше 0.
* скорее всего, он не превысит 100 лет, но, учитывая то, что нашей программой могут пользоваться и долгожители (если верить статистике, некоторые доживают и до 150) - переспросим пользователя, уверен ли он в том, что ввел именно это значение, и, если он согласится, примем его.

Так же мы ограничим количество возможных попыток для ввода возраста тремя.

Добавим в документ Microsoft *Word* две кнопки. Одну из них назовем cmd\_AgeInput и подпишем как Введите ваш возраст, вторую назовем cmd\_InpAge\_Easy и подпишем как Простой ввод возраста.

Для проверок создадим несколько пользовательских функций, реализующих их. Такой подход очень удобен, так как позволяет уменьшить количество сложночитаемых вложенных конструкций If-Then-Else и делает работу с программой и ее *отладку* проще.

В [листинге 8.2](https://www.intuit.ru/studies/courses/494/350/lecture/8364?page=3#example.8.2). приведена *функция*, которая проверяет, соответствует ли введенное число нашим требованиям.

Public Function IsGoodNum(var\_Inp) As Boolean

'Функция возвращает True, если введенное

'число подходит, иначе - False

If IsNumeric(var\_Inp) = True \_

And Val(var\_Inp) > 0 \_

And Val(var\_Inp) < 255 \_

Then

IsGoodNum = True

Else

IsGoodNum = False

End If

End Function

Листинг 8.2. Функция для проверки введенного числа

В [листинге 8.3](https://www.intuit.ru/studies/courses/494/350/lecture/8364?page=3#example.8.3) вы можете найти функцию, которая проверяет введенное число и возвращает True, если оно больше 100.

Public Function IsMore100(var\_Inp) As Boolean

'Функция возвращает True, если введенное число

'более 100

If Val(var\_Inp) > 100 Then

IsMore100 = True

Else

IsMore100 = False

End If

End Function

Листинг 8.3. Функция для выяснения, больше 100 введенное число или нет

В [листинге 8.4](https://www.intuit.ru/studies/courses/494/350/lecture/8364?page=3#example.8.4). вы можете найти код функции, которая возвращает строку с описанием ошибки, которую допустил *пользователь* при вводе возраста.

Public Function NotNumber(var\_Inp) As String

'Определяет, что именно ввел пользователь

'вместо ожидаемого числа и возвращает

'строку для вывода сообщения об ошибке

'Если не выполнится ни одно из условий

'программа выдаст общее сообщение об

'ошибке ввода данных

NotNumber = "Ошибка ввода данных"

'Если ей удастся определить тип ошибки

'будет выведено более точное сообщение

If var\_Inp = "" Then

NotNumber = "Вы забыли ввести возраст"

End If

If Val(var\_Inp) < 0 Then

NotNumber = "Вы ввели отрицательное число"

End If

If IsNumeric(var\_Inp) = False Then

NotNumber = "Вы ввели не число"

End If

If Val(var\_Inp) > 255 Then

NotNumber = "Вы ввели слишком большое число"

End If

End Function

Листинг 8.4. Функция для выяснения ошибки пользователя

[Листинг 8.5](https://www.intuit.ru/studies/courses/494/350/lecture/8364?page=3#example.8.5) содержит текст программы, которая использует вышеописанные функции для проверки введенного пользователем возраста.

Private Sub cmd\_AgeInput\_Click()

'Переменная для ввода пользователм

Dim str\_Inp

'Сюда запишем возраст после проверки

Dim num\_Age As Byte

'А здесь будем хранить ответ пользователя на

'Вопрос о том, уверен ли он в том, что ему

'Больше 100 лет

Dim IsYouMore100 As Variant

'В этой переменной будем хранить количество

'Неправильных попыток ввода

'Если оно превысит 3 - выходим из программы

Dim num\_InpTry As Byte

'Эта переменная нужна для того, чтобы

'определить, окончен ли опрос пользователя

Dim bool\_Flag As Boolean

bool\_Flag = False

'Этот цикл будет длиться до тех пор,

'пока пользователь не введет верное значение

'или пока не совершит максимальное число

'попыток

Do

str\_Inp = InputBox("Введите ваш возраст")

If num\_InpTry > 3 Then

MsgBox "Вы совершили более 3 ошибок"

Exit Sub

End If

num\_InpTry = num\_InpTry + 1

'Если введенное число подходит

If IsGoodNum(str\_Inp) = True Then

'Проверяем - не больше ли оно 100

If IsMore100(str\_Inp) = False Then

'Если не больше - просто устанавливаем

'флаг выхода в True

'и записываем возраст в переменную num\_Age

bool\_Flag = True

num\_Age = CByte(str\_Inp)

Else

'Если возраст больше 100 - переспросим

IsYouMore100 = \_

MsgBox("Вам точно больше 100?", vbYesNo)

'Если пользователь не ошибся,

'Установим флаг выхода

'и запишем возраст в num\_Age

If IsYouMore100 = vbYes Then

bool\_Flag = True

num\_Age = CByte(str\_Inp)

End If

End If

Else

'Если пользователь изначально ввел

'неподходяще число,

'сгенерируем с помощью функции NotNumber

'сообщение об ошибке и выведем его

MsgBox (NotNumber(str\_Inp))

End If

Loop Until bool\_Flag = True

MsgBox ("Ваш возраст: " + Str(num\_Age))

End Sub

Листинг 8.5. Программа, запрашивающая у пользователя возраст и проверяющая его с помощью вышеописанных функций

Как видите, обычная проверка возраста пользователя оказывается не слишком простым делом. Однако серьезная *программа* не может обойтись без подобных проверок. Конечно же, возможны варианты проверки. Например, чтобы проверить возраст, и если он нас не устраивает, без дополнительных сообщений предложить пользователю снова ввести его, достаточно такой конструкции ([листинг 8.6](https://www.intuit.ru/studies/courses/494/350/lecture/8364?page=3#example.8.6).):

Private Sub cmd\_InpAge\_Easy\_Click()

Dim str\_Inp

Dim num\_Age As Byte

'Цикл продолжается до тех пор,

'пока введенное значение не станет

'числом, находящимся в диапазоне от

'0 (не включая 0) до 100

Do

str\_Inp = \_

InputBox("Введите ваш возраст")

Loop Until \_

IsNumeric(str\_Inp) = True \_

And Val(str\_Inp) <= 100 \_

And Val(str\_Inp) > 0

'Конвертируем введенное число в

'тип данных Byte

num\_Age = CByte(Val(str\_Inp))

MsgBox ("Ваш возраст: " + Str(num\_Age))

End Sub

Листинг 8.6. Простая проверка введенного возраста

Как видите, здесь мы ограничились тройной проверкой на выходе из *цикла с постусловием*. А именно, проверили, является ли введенное *значение* числом, и попадает ли оно в *диапазон* 0-100, исключая 0.

**8.9. Работа с ошибками времени выполнения**

[08-03-Ошибки времени выполнения.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/8/files/08_03.docm) - пример к п. 8.9.

Выше мы рассмотрели метод борьбы с ошибками, который заключался в их предотвращении. Но что делать, если в ходе работы программы происходит ошибка, для предотвращения которой ничего сделано не было?

В этой ситуации нам поможет перехват и обработка *ошибок времени выполнения*.

Давайте рассмотрим ситуацию, которая гарантированно вызовет ошибку - *деление* на 0. Добавим в документ Microsoft *Word* кнопку с именем cmd\_SimpleInput, подпишем ее как Отсутствие перехвата ошибок.

Напишем простую программу (поместив ее в обработчик события Click кнопки cmd\_SimpleInput ), которая запрашивает у пользователя два числа, делит одно на другое и выводит результат деления. Следующий код ([листинг 8.7](https://www.intuit.ru/studies/courses/494/350/lecture/8364?page=4#example.8.7).) реализует эту функциональность:

Dim num\_A

Dim num\_B

Dim num\_C

num\_A = InputBox("Введите число A")

num\_B = InputBox("Введите число B")

num\_C = num\_A / num\_B

MsgBox ("А, деленое на В равняется: " + Str(num\_C))

Листинг 8.7. Деление без обработки ошибок

Чтобы организовать обработку ошибок в программе нужно расположить перед потенциально опасным участком (в нашем случае - перед строкой, где происходит *деление*) оператор такого вида:

On *Error* GoTo Идентификатор\_обработчика

*Идентификатор* обработчика ошибок - это номер или (что предпочтительнее) *метка* строки в коде текущей процедуры, куда должно быть передано управление при возникновении ошибки. Обработчик ошибок должен находиться в текущей процедуре. Использовать в качестве идентификатора обработчика имя процедуры или функции нельзя

В обработчике можно исправить ошибку, предложить пользователю выполнить какие-то действия или просто вывести *сообщение об ошибке* и завершить работу программы. В обработчике ошибок можно использовать оператор *Resume* - он возвращает управление в строку, в которой произошла ошибка.

После того, как опасный участок программы пройден, обработку ошибок желательно отключить - это можно сделать такой командой:

On Error GoTo 0

Так же при обработке ошибок можно пользоваться командой

On Error Resume Next

Эта *команда* подавляет *вывод* сообщения об ошибке, просто передавая управление в следующую, после ошибочной, строку.

Для работы с ошибками, возникающими при выполнении программы, существует специальный *объект* - *Err*. Он содержит информацию об ошибках *времени выполнения*. У этого объекта есть несколько важнейших методов и свойств

Метод Clear очищает информацию об ошибках в объекте *Err*.

Метод Raise позволяет сгенерировать ошибку, указав в качестве параметра ее номер и некоторые другие, необязательные, данные. Эта возможность полезна при тестировании программ, оснащенных обработчиками ошибок, чтобы разработчик мог произвольно генерировать те или иные ошибки.

Номера ошибок находятся в диапазоне 0 - 65535. В частности, номера 0 - 512 зарезервированы за системными ошибками, 513 - 65535 - за ошибками, коды которых *пользователь* может назначать самостоятельно.

Например, для вызова ошибки *несоответствия типов* переменных (она возникает при попытке записать в числовую переменную строковое *значение* и в других случаях), которая имеет код 13, достаточно выполнить такую команду:

Err.Raise (13)

Свойство Number содержит номер ошибки. Обычно этот номер используют для выбора действия в обработчике ошибок.

Свойство Description содержит текстовое описание ошибки

Научим нашу программу обрабатывать ошибки, которые могут возникать при вводе "неправильных" чисел. В нашем простом примере возможно возникновение как *минимум* трех различных ошибок

* если число B (делитель) будет равно 0 - это ошибка №11 - "Деление на 0"
* если одно из введенных чисел (или оба) на самом деле - не числа. Например, пользователь случайно нажал OK, не введя число в поле ввода. Это - ошибка №13 - "Несоответствие типов"
* если оба числа равны нулю - такая ситуация вызовет ошибку №6 - "Переполнение"

Нужно учитывать так же, что чисто теоретически возможно возникновение ошибки, для которой мы не предусмотрели определенный раздел обработчика.

В результате этих рассуждений у нас получился такой код ([листинг 8.8](https://www.intuit.ru/studies/courses/494/350/lecture/8364?page=4#example.8.8).), который мы разместим в обработчике события Click кнопки cmd\_ErrHandler, подписанной как Обработка ошибок и размещенной в том же документе, что и вышеописанная кнопка

Dim num\_A

Dim num\_B

Dim num\_C

num\_A = InputBox("Введите число A")

num\_B = InputBox("Введите число B")

On Error GoTo ErrHandler

'С этого момента начинаем наблюдение

'за ошибками

num\_C = num\_A / num\_B

On Error GoTo 0

'Прекращаем наблюдение за ошибками

MsgBox ("А, деленое на В равняется: " + \_

Str(num\_C))

Exit Sub

ErrHandler: 'Начало обработчика ошибок

MsgBox ("Вызвана ошибка №" + \_

Str(Err.Number) + " " + \_

Err.Description)

Select Case Err.Number

Case 6:

'Overflow - переполнение

'обе переменные равны 0

MsgBox ("Вы ввели A и B = 0")

num\_A = InputBox("Введите число A")

num\_B = InputBox("Введите число B")

Case 11:

'Деление числа на 0

MsgBox ("Вы ввели B = 0")

num\_B = InputBox("Введите число B")

Case 13:

'Type mismatch - несоответствие типов

'одно из чисел или оба - не числа

MsgBox ("Одно из чисел или оба - не числа")

num\_A = InputBox("Введите число A")

num\_B = InputBox("Введите число B")

Case Else:

'на случай возникновения

'непредусмотренной ошибки

num\_A = InputBox("Введите число A")

num\_B = InputBox("Введите число B")

End Select

Resume

End Sub

Листинг 8.8. Обработка ошибок

Обратите внимание на то, что перед меткой строки, с которой начинается обработчик мы поставили команду Exit Sub. Это сделано для того, чтобы *программа*, дойдя до последней строки своей основной части, не начала бы выполнять обработчик ошибок.

Перед опасной операцией деления мы поставили команду On Error GoTo ErrHandler - если при делении произойдет ошибка, будет вызван обработчик. Необязательно устанавливать эту команду строго перед опасной процедурой - мы можем поставить ее в любом месте программы до нее. После того, как опасный участок пройден - следует *команда* On Error GoTo 0 - она отключает *слежение* за ошибками.

Внутри обработчика мы, прежде всего, выводим окно сообщения, где указываем номер ошибки и описание - и то и другое получено с помощью соответствующих свойств объекта *Err*. Далее, мы используем номер, хранящийся в свойстве Number объекта *Err*, чтобы произвести различные действия в зависимости от ошибки. В частности, наша *программа* может определять три вида ошибок ввода, сообщать об этом пользователю и запрашивать ввод переменных. Для обработки других ошибок, которые могут возникнуть, создана *ветвь* Case Else.

Как видите, обработка *ошибок времени выполнения* - задача непростая. Нужно определить, какие именно ошибки могут возникнуть, написать код обработчика для каждой из них. Однако, если вы хотите, чтобы ваши программы выглядели и работали профессионально - без работы над ошибками не обойтись.

**8.10. Выводы**

В этой лекции мы рассмотрели вопросы *отладки программ* и работы с ошибками. Теперь вы владеете основными приемами *VBA*, общими для программирования в любых приложениях Office. В следующих лекциях мы займемся программированием для MS *Word* и MS *Excel*.

Лекция 9. Объектная модель MS Word, объект Application

**9.1. Объектная модель MS Word**

Microsoft *Word* 2007, как, впрочем, и другие приложения Office, имеет большую *объектную модель*, в которую входят множество объектов и коллекций. Однако на практике чаще всего используются лишь некоторые из них.

В частности, это следующие:

* Word (Microsoft Word) - чтобы вызывать объекты Microsoft Office Word из других приложений используется объект Word.
* Application (Приложение) - объект, который включает в себя все остальные объекты и коллекции. Сам объект Application включается в объект Word.
* Documents (Документы) - коллекция, которая включает в себя объекты Document (Документ). Объект Document представляет собой документ, с которым вы работаете в редакторе.
* *Bookmarks* (Закладки) - c помощью объектов этой коллекции можно задавать места в документе, в которые, при автоматическом создании, можно вставлять изменяемые части.
* Paragraphs (Абзацы), *Sentences* (Предложения), Words (Слова), Characters (Символы), Tables (Таблицы), OMath (Математические формулы), Эти коллекции, содержащие объекты, соответствующие их названиям, могут использоваться для обработки слов документа ( Words ), предложений ( *Sentences* ), отдельных символов ( Characters ) и т.д.
* Selection (Выделенная область) - представляет собой выделенную область документа или позицию, в которую будет осуществляться вставка *очередного символа*. Этот объект часто используют при создании простых документов. Однако у него есть существенные недостатки, что делает предпочтительнее использование объекта Range в большинстве случаев.
* Range (Диапазон) - входит в Document и другие объекты, представляет собой диапазон документа, ограниченный начальным и конечным символом.

Работая с *объектной моделью MS Word* следует понимать, что, например, объекты Document и Selection могут содержать одинаковые коллекции объектов, например, и там и там есть коллекция Characters, в которую входят объекты Character - то есть - символы. Однако в случае с объектом Document мы можем работать с любым символом документа, а в случае с Selection - лишь с символами в пределах выделенной области. Однако, несмотря на различия, и там и там коллекция Characters имеет одинаковые свойства и методы. То же самое касается и других коллекций и объектов.

Давайте начнем с объекта Application и здесь же поговорим об использовании *объектных переменных*.

**9.2. Работа с объектными переменными**

***Объектные переменные - это переменные, которые хранят ссылки на объекты***. Чтобы инициализировать *объектную переменную*, нужно использовать *ключевое слово* Set. Например, так ([листинг 9.1](https://www.intuit.ru/studies/courses/494/350/lecture/8366?page=1#example.9.1).):

Set obj\_NewWord = Word.Application

Листинг 9.1. Связывание объекта с объектной переменной

Здесь вы можете видеть *присваивание* переменной ссылки на *объект* Word.Application. Код, подобный этому, нужен в других приложениях Microsoft Office для запуска нового экземпляра MS *Word*.

Для того чтобы вы могли работать с Microsoft *Word* из других приложений - подключите библиотеку Microsoft Word 12.0 *Object Library*. Сделать это можно, открыв в редакторе окно *References* командой Tools o References.

Немного ниже мы поговорим о том, что можно делать с *объектными переменными*, а пока сосредоточимся на их объявлении и инициализации.

Как видите, в [листинге 9.1](https://www.intuit.ru/studies/courses/494/350/lecture/8366?page=1#example.9.1) мы не объявляли переменную - мы сразу присвоили ей ссылку на *объект*. При последующей работе с такой необъявленной *объектной переменной*, мы не сможем пользоваться подсказкой по свойствам и методам. Вспомните - когда вы набираете в редакторе имя элемента управления и ставите после него точку - вы видите подсказку. Это очень удобно, так как позволяет избежать ошибок и излишних "походов" в справочную систему *VBA*.

Для того чтобы справка по свойствам и методам работала, *объектную переменную* надо сначала объявить, а потом присвоить ей ссылку на *объект*. Например, так ([листинг 9.2](https://www.intuit.ru/studies/courses/494/350/lecture/8366?page=1#example.9.2).):

Dim obj\_NewWord As Word.Application

Set obj\_NewWord = Word.Application

Листинг 9.2. Предварительное объявление объектной переменной

Ссылку на *объект* можно присвоить в процессе объявления переменной. Для этого нужно воспользоваться ключевым словом New ([листинг 9.3](https://www.intuit.ru/studies/courses/494/350/lecture/8366?page=1#example.9.3).):

Dim obj\_NewWord As New Word.Application

Листинг 9.3. Ключевое слово New при объявлении переменной

Переменную, объявленную таким образом, мы можем сразу же использовать по назначению. Новый экземпляр объекта будет создан при первом обращении к нему.

О том, как же пользоваться *объектными переменными*, скорее даже не ими, а объектами, на которые они ссылаются, мы поговорим ниже.

**9.3. Объект Application - приложение**

[09-01-With For Each.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/9/files/09_01.docm) - пример к п. 9.3.

*Объект* Application можно представить в виде приложения Microsoft *Word*.

Такой код позволит вывести в окно сообщения информацию об имени приложения:

MsgBox Application.Name

Как видите, здесь мы обратились к свойству Name (Имя) объекта Application. Вот, что будет выведено при его исполнении ([рис. 9.1](https://www.intuit.ru/studies/courses/494/350/lecture/8366?page=1#image.9.1).).



**Рис. 9.1.**Свойство Name объекта Application

Здесь мы обходимся без *объектных переменных*, работая с уже существующим объектом. Ведь если этот код исполняется в Microsoft *Word*, это значит, что сам Microsoft *Word* уже запущен.

Чтобы упростить обращение к нескольким свойствам объекта, можно воспользоваться оператором With-End With. Этот оператор позволяет обращаться к нескольким свойствам или методам объекта в упрощенной форме. Например, чтобы вывести имя приложения и узнать номер сборки программы нужно воспользоваться таким построением ([листинг 9.4](https://www.intuit.ru/studies/courses/494/350/lecture/8366?page=1#example.9.4).):

With Application

MsgBox .Name

MsgBox .Build

End With

Листинг 9.4. Оператор With - End With

Здесь мы использовали *объект* Application - вместо него может быть любой другой *объект* или *объектная переменная*.

Еще один оператор, которым удобно пользоваться при работе с объектами и коллекциями - это For Each…Next.

Например, для вывода в окнах сообщений имен всех открытых документов, можно написать такой код ([листинг 9.5](https://www.intuit.ru/studies/courses/494/350/lecture/8366?page=1#example.9.5).):

Dim var\_Doc

For Each var\_Doc In Application.Documents

MsgBox var\_Doc.Name

Next var\_Doc

Листинг 9.5. Оператор For Each - Next

Var\_Doc - это *переменная* типа Variant. Коллекция Application.Documents содержит все открытые документы. При каждом проходе *цикла* в переменную var\_Doc записывается *ссылка* на очередной *объект* в коллекции.

Чтобы выйти из *цикла*, можно воспользоваться оператором Exit For.

Сейчас мы кратко опишем важнейшие *методы*, *свойства* и *события объекта* Application.

**9.4. Методы объекта Application**

**9.4.1. BuildKeyCode, KeyString - горячие клавиши**

BuildKeyCode возвращает уникальный цифровой код для заданной комбинации клавиш. Используется при назначении "горячих клавиш" для выполнения различных операций.

KeyString возвращает комбинацию клавиш для переданного кода.

Ниже мы рассмотрим пример программного назначения клавиатурной комбинации для вызова макроса.

**9.4.2. ChangeFileOpenDirectorу - путь для работы с файлами**

ChangeFileOpenDirectory позволяет установить директорию, в которой Microsoft Word ищет файлы для открытия. В установленной папке программа будет искать файлы при программном открытии файлов и при вызове окна открытия файлов.

Выбранная директория будет использоваться для открытия файлов по умолчанию лишь до тех пор, пока Word не будет перезапущен.

Например, чтобы установить папку " C:\Новые документы " в качестве папки по умолчанию, можно использовать такой код ([листинг 9.6](https://www.intuit.ru/studies/courses/494/350/lecture/8366?page=1#example.9.6).):

Application.ChangeFileOpenDirectory ("C:\Новые документы")

Листинг 9.6. Установка новой папки по умолчанию

Следует отметить, что если мы вызываем *метод* или *свойство объекта* Application из макроса Microsoft Word, мы можем опускать вызов Application - то есть, в вышеприведенном примере можно написать так:

ChangeFileOpenDirectory ("C:\Новые документы")

**9.4.3. CleanString - очистка строк**

CleanString очищает переданную строку от непечатаемых и специальных символов, превратив ее в обычный текст. Такой же текст можно получить, если скопировать текст из Microsoft Word в Блокнот, а потом - обратно. Например, с помощью такого кода можно получить очищенную строку из выделенной области документа ([листинг 9.7](https://www.intuit.ru/studies/courses/494/350/lecture/8366?page=1#example.9.7).).

str\_Clean = Application.CleanString(Selection.Text)

Листинг 9.7. Очистка строки

**9.4.4. Keyboard - программное переключение раскладки**

*Keyboard* позволяет программно переключать раскладку клавиатуры. При переключении используется идентификатор языковых и клавиатурных установок. Чтобы включить русскую раскладку, можно воспользоваться таким кодом ([листинг 9.8](https://www.intuit.ru/studies/courses/494/350/lecture/8366?page=1#example.9.8).):

Application.Keyboard 1049

Листинг 9.8. Переключение на русскую раскладку клавиатуры

Очевидно, что 1049 - это код русского языка. Для включения английской раскладки используйте этот метод с идентификатором 1033.

**9.4.5. ListCommands - справка по горячим клавишам**

ListCommands это необычный метод - если вызвать его с параметром True ([листинг 9.9](https://www.intuit.ru/studies/courses/494/350/lecture/8366?page=1#example.9.9).) - он создаст новый документ, содержащий таблицу со списком клавиатурных сочетаний, назначенных командам MS Word. Таблица содержит несколько десятков страниц.

Application.ListCommands True

Листинг 9.9. Вывод информации о горячих клавишах

**9.4.6. NewWindow - копия окна активного документа**

[09-02-New Window.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/9/files/09_02.docm) - пример к п. 9.4.6.

NewWindow создает копию окна активного документа. Это может быть полезным например, для того, чтобы вывести один и тот же документ на два монитора при работе с расширенным рабочим столом и т.д. Например, метод можно вызвать так:

Application.NewWindow

Листинг 9.10. Создание копии окна активного документа

При необходимости можно задать, для какого именно документа вы хотите создать копию окна. Однако, это потребует использования других объектов. Например - коллекции Windows, содержащей окна документов. Такой код ([листинг 9.11](https://www.intuit.ru/studies/courses/494/350/lecture/8366?page=1#example.9.11).) создаст одну копию для каждого открытого окна:

For i = Application.Windows.Count To 1 Step -1

Application.Windows.Item(i).NewWindow

Next i

Листинг 9.11. Создание копии для каждого открытого окна

Мы не случайно запустили цикл обхода открытых окон в обратном порядке. Дело в том, что иначе будет создано несколько копий одного и того же документа.

В этом примере вам встретились неизвестные ранее свойства Count и Item. Первое обычно содержит количество элементов в коллекции, второе позволяет обращаться к элементам по индексам или именам.

При создании копии после имени документа появляется двоеточие с номером окна, например, ":1" для первого, ":2" для второго и т.д.

**9.4.7. OnTime - запуск макросов по расписанию**

[09-03-OnTime.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/9/files/09_03.docm) - пример к п. 9.4.7.

OnTime позволяет устанавливать таймер, по которому можно запустить макрос в определенное время. Например, чтобы выполнить макрос MyMacros, который хранится в том же документе, что и исполняемый макрос, достаточно выполнить такой код ([листинг 9.12](https://www.intuit.ru/studies/courses/494/350/lecture/8366?page=1#example.9.12).).

Application.OnTime When:="08:37:00", Name:="MyMacros"

Листинг 9.12. Установка таймера запуска макроса

Параметр When указывает на время запуска, параметр Name - на имя макроса. Важно, чтобы макрос был доступен как в момент запуска таймера, так и в момент запуска самого макроса. Для того чтобы это условие соблюдалось, желательно хранить макросы, запускаемые по таймеру, в Normal.dotm, в другом глобальном шаблоне, или в том же документе, в котором произошел запуск таймера.

Как видите, мы привели пример лишь для запуска макроса в строго определенное время. А как же быть, если нужно запустить макрос, например, через пятнадцать секунд после выполнения какой-либо процедуры? Ответ прост ([листинг 9.13](https://www.intuit.ru/studies/courses/494/350/lecture/8366?page=1#example.9.13).)

Application.OnTime \_

When:=Now + TimeValue("00:00:15"), \_

Name:="MyMacros"

Листинг 9.13. Запуск макроса по расписанию

Здесь мы запускаем тот же самый макрос, но уже через 15 секунд после установки таймера. Функция Now имеет тип Date и возвращает текущую дату и время, а функция TimeValue преобразует переданное ей время в формат Date.

Если вы хотите, чтобы какой-то макрос запускался с некоторой периодичностью - установку таймера на запуск этого макроса можно установить в нем. Первый запуск макроса можно произвести вручную или каким-то другим способом (например - по событию приложения, с помощью автоматически выполняемого макроса), а последующие запуски этот макрос будет инициировать самостоятельно.

**9.4.8. PrintOut - распечатка документов**

*PrintOut* предназначен для вывода документов на печать. Например, следующий код ([листинг 9.14](https://www.intuit.ru/studies/courses/494/350/lecture/8366?page=2#example.9.14).) выводит на печать все документы Microsoft Word 2007 (то есть - с расширениями docx и docm ), расположенные в корневом каталоге диска C.

var\_Doc = Dir("C:\\*.doc?")

Do While var\_Doc <> ""

Application.PrintOut FileName:=var\_Doc

var\_Doc = Dir()

Loop

Листинг 9.14. Печать всех документов из корневого каталога диска C

Метод *PrintOut* существует для различных объектов и может принимать множество параметров, управляющих всеми тонкостями печати. Например, с помощью такого кода ([листинг 9.15](https://www.intuit.ru/studies/courses/494/350/lecture/8366?page=2#example.9.15).) мы можем распечатать первые пять страниц текущего документа:

ActiveDocument.ActiveWindow.PrintOut \_

Range:=wdPrintFromTo, From:="1", To:="5"

Листинг 9.15. Вывод на печать первых 5 страниц текущего документа

**9.4.9. Quit - выход из приложения**

Quit используется для выхода из приложения ([листинг 9.16](https://www.intuit.ru/studies/courses/494/350/lecture/8366?page=2#example.9.16).):

Application.Quit

Листинг 9.16. Выход из приложения

**9.4.10. Run - запуск макросов**

Run позволяет запускать макросы и, при необходимости, передавать им параметры (до 30). При вызове методу Run передается имя, состоящее из имени проекта, имени модуля, и, собственно, имени макроса. Например, запуск макроса MyMacros, расположенного в модуле NewModule проекта Project1 выглядит так ([листинг 9.17](https://www.intuit.ru/studies/courses/494/350/lecture/8366?page=2#example.9.17).):

Application.Run ("Project1.NewModule.MyMacros")

Листинг 9.17. Запуск макроса с помощью метода Run

Параметры, передаваемые макросу, перечисляются после его имени через запятую.

**9.4.11. ScreenRefresh - принудительное обновление экрана**

ScreenRefresh позволяет принудительно обновить экран. Используется обычно в комбинации с запретом автоматического обновления экрана (свойство Application.ScreenUpdating, его мы рассмотрим ниже). Как правило, обновление экрана делают через некоторые промежутки времени - тогда у пользователя не возникает ощущения, что программа зависла.

Теперь рассмотрим наиболее полезные свойства объекта Application.

**9.5. Свойства объекта Application**

**9.5.1. ActiveDocument и другие - активный документ**

ActiveDocument возвращает объект активного документа - того, который открыт в данный момент в Microsoft Word для редактирования. У объекта ActiveDocument есть множество полезных свойств и методов. Мы обсудим их при разговоре об объекте Document.

Например, в [листинге 9.18](https://www.intuit.ru/studies/courses/494/350/lecture/8366?page=2#example.9.18) мы выводим в окне сообщения имя активного документа.

MsgBox Application.ActiveDocument.Name

Листинг 9.18. Вывод имени активного документа

Надо отметить, что нельзя модифицировать свойство ActiveDocument - то есть не можем с его помощью сделать неактивный документ активным. Чтобы сделать документ активным, используют специальный метод объекта Document.

Существует немало других свойств Application, имена которых начинаются с Active.

ActivePrinter возвращает объект активного принтера - устройства, используемого для печати документов по умолчанию.

ActiveWindow возвращает активное окно - объект типа Window.

**9.5.2. CapsLock - текущее состояние Caps Lock**

CapsLock показывает текущее состояние режима *Caps Lock*. Если этот режим активен, по умолчанию при вводе с клавиатуры вводятся прописные буквы. Если режим *Caps Lock* включен - свойство возвращает True, если выключен - False.

**9.5.3. Caption - заголовок окна MS Word**

Caption позволяет узнать, и, при желании, изменить заголовок окна Microsoft Word.

Например код в листинге 9.19. меняет словосочетание Microsoft Word в заголовке окна на слова "Моя программа".

Application.Caption = "Моя программа"

Листинг 9.19. Изменение заголовка окна MS Word

**9.5.4. CustomizationContext - область сохранения настроек**

CustomizationContext позволяет узнавать и задавать документ и шаблон, в котором сохраняются такие настройки, как изменения в настройке меню, панелей инструментов, горячих клавиш. Чтобы узнать текущую область настройки, достаточно выполнить код из листинга 9.20.

MsgBox Application.CustomizationContext

Листинг 9.20. Узнаем область сохранения настроек

Чтобы установить в качестве области настройки шаблон Normal.dotm (то есть - сделать настройки доступными для всех документов), достаточно воспользоваться кодом листинга 9.21.:

CustomizationContext = NormalTemplate

Листинг 9.21. Будем сохранять изменения настроек в Normal.dotm

Обратите внимание на то, что в предыдущих версиях MS Word этот шаблон назывался Normal.dot.

Для сохранения изменений в шаблоне, присоединенном к активному документу, можно воспользоваться командой из листинга 9.22.:

CustomizationContext = ActiveDocument.AttachedTemplate

Листинг 9.22. Сохраняем изменения в присоединенном шаблоне

Для сохранения изменений лишь в текущем документе, можно использовать команду из листинга 9.23.

CustomizationContext = ThisDocument

Листинг 9.23. Сохраняем изменения в присоединенном шаблоне

**9.5.5. Dialogs - диалоговые окна MS Word**

[09-04-Dialogs.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/9/files/09_04.docm) - пример к п. 9.5.5.

Dialogs возвращает коллекцию Dialogs (Диалоговые окна), которая дает доступ ко всем диалоговым окнам Microsoft Word. Если выполнить код листинга 9.24., можно узнать количество объектов в коллекции Dialogs.

MsgBox ("Количество окон в коллекции Dialogs:" + \_

Str(Application.Dialogs.Count))

Листинг 9.24. Количество диалоговых окон в коллекции Dialogs

Чтобы отобразить диалоговое окно поиска строк в документе, можно использовать код, представленный в листинге 9.25.

Dim obj\_FindS As Dialog

Set obj\_FindS = Dialogs(wdDialogEditFind)

With obj\_FindS

.Find = "Строка"

.Show

End With

Листинг 9.25. Запуск диалогового окна поиска строк в документе

Здесь мы объявляем *объектную переменную* типа Dialog, создаем ссылку на окно поиска строк в документе - это окно представлено константой wdDialogEditFind, после чего присваиваем свойству Find окна поиска значение "Строка" - именно это значение будет отображаться в строке поиска, и показываем окно поиска, используя метод Show. Похожим образом работают и с другими диалоговыми окнами. Их имена (около 230) можно найти в перечислении WdWordDialog.

**9.5.6. DisplayAlerts - управление выводом сообщений**

DisplayAlerts - позволяет управлять выводом диалоговых окон, сообщений об ошибках и т.д. при работе в Microsoft Word. Чтобы запретить вывод всех сообщений, используйте код, приведенный в листинге 9.26.

Application.DisplayAlerts = wdAlertsNone

Листинг 9.26. Запрет вывода всех сообщений

Константа wdAlertsNone запрещает вывод сообщений. Аналогично, константа wdAlertsAll разрешает вывод всех сообщений, а wdAlertsMessageBox разрешает лишь вывод окон сообщений, сообщения об ошибках не выводятся.

**9.5.7. EnableCancelKey - запрещаем остановку программы**

[09-05-Cancel.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/9/files/09_05.docm) - пример к п. 9.5.7

EnableCancelKey - позволяет разрешать и запрещать пользователю остановку программы по нажатию сочетания клавиш Ctrl + Break. Это может быть полезно при выполнении участков кода, которые нельзя прерывать.

Для того чтобы запретить прерывание работы программы нужно присвоить этому свойству значение wdCancelDisabled, для разрешения - wdCancelInterrupt.

Пример использования этого оператора вы можете найти в листинге 9.27.

Dim num\_b As Double

MsgBox ("Вы не сможете прервать выполнение " + \_

"следующего кода")

Application.EnableCancelKey = wdCancelDisabled

For i = 1 To 1000000

For j = 100 To 1 Step -1

num\_b = num\_b \* i / j

Next j

Next i

Application.EnableCancelKey = wdCancelInterrupt

MsgBox ("А теперь Ctrl + Pause Break " + \_

"снова работает")

Листинг 9.27. Запрет прерывания выполнения программы

**9.5.8. IsObjectValid - проверка объектных переменных**

IsObjectValid - позволяет проверить *объектную переменную*. Если объект, на которую она ссылается, существует - проверка возвратит True, если нет - False. Это свойство полезно использовать для проверки объектов, которые могут быть удалены пользователем. Если объект существует, можно произвести с ним какие-либо действия. Если нет - сообщить причину, по которой действия невозможны.

**9.5.9. KeyBindings - назначаем клавиатурные сокращения**

KeyBindings - возвращает коллекцию KeyBindings, которая содержит информацию о клавиатурных привязках. Это очень полезная коллекция - с ее помощью можно, например, назначить клавиатурную комбинацию для запуска какого-нибудь макроса. [листинг 9.28](https://www.intuit.ru/studies/courses/494/350/lecture/8366?page=2#example.9.28). позволяет назначить комбинацию клавиш Alt + Shift + T макросу TextEdit, хранящемуся в модуле MyMacros шаблона Normal.Dotm

Application.CustomizationContext = NormalTemplate

Application.KeyBindings.Add \_

wdKeyCategoryMacro, \_

"Normal.MyMacros.TextEdit", \_

BuildKeyCode(wdKeyAlt, wdKeyShift, wdKeyT)

Листинг 9.28. Программное назначение комбинации клавиш макросу

Сначала с помощью свойства CustomizationContext мы устанавливаем место, где будет сохранена привязка. Это - шаблон Normal.Dotm. Далее мы используем метод Add коллекции KeyBindings. Мы передаем этому методу три параметра. Первый ( wdKeyCategoryMacro ) указывает методу на то, что мы назначаем клавиатурную комбинацию макросу. Второй -" Normal.MyMacros.TextEdit " - указывает путь к макросу, запуск которого мы назначаем клавиатурному сокращению. Третий параметр содержит вызов метода BuildKeyCode - напомню, что он генерирует код сочетания клавиш на основе переданных ему параметров. В нашем случае он сгенерирует код для сочетания клавиш Alt + Shift + T - мы передали методу параметры wdKeyAlt, wdKeyShift и wdKeyT.

Если все сделано верно - в частности, макрос, запуск которого мы автоматизируем, существует - после выполнения такого кода нажатие выбранного сочетания клавиш в любом активном документе приведет к запуску этого макроса.

**9.5.10. NumLock - состояние цифровой клавиатуры**

NumLock возвращает состояние клавиши NumLock - True если цифровая клавиатура находится в *режиме ввода* цифр, иначе - False.

**9.5.11. RecentFiles - недавно открытые файлы**

RecentFiles - позволяет работать с файлами, которые вы недавно открывали. Список этих файлов можно увидеть в диалоговом окне открытия файлов. Последний файл, с которым вы работали, хранится в списке первым. Чтобы открыть его, можно воспользоваться кодом из листинга 9.29.

RecentFiles(1).Open

Листинг 9.29. Открываем последний из недавно открытых файлов

**9.5.12. ScreenUpdating - запрет обновления экрана**

[09-06-ScreenUpdating.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/9/files/09_06.docm) - пример к п. 9.5.12.

Свойство ScreenUpdating используют для отключения обновления экрана во время вывода в документ большого количества информации или других действий с документом. Это позволяет ускорить работу, так как системные ресурсы не тратятся на постоянное обновление экрана. В листинге 9.30. мы запрещаем обновление экрана, выводим в документ 10000 строк, принудительно обновляя экран после каждой 1000 строк, после чего разрешаем автоматическое обновление.

Application.ScreenUpdating = False

For i = 1 To 10

For j = 1 To 1000

Selection.TypeText ("Строка №" + Str(i \* j))

Selection.TypeParagraph

Next j

Application.ScreenRefresh

Next i

Application.ScreenUpdating = True

Листинг 9.30. Запрет и разрешение обновления экрана

Очевидно, что присвоив False свойству ScreenUpdating мы запрещаем обновление экрана, а присвоив True - разрешаем.

**9.5.13. StatusBar - вывод информации в строку состояния**

[09-07-Status Bar.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/9/files/09_07.docm) - пример к п. 6.3.2.13

Модифицировав свойство StatusBar вы можете записать любой текст в строку состояния Microsoft Word. Туда можно выводить сообщения о ходе выполнения какой-нибудь *длительной операции* и т.д. В листинге 9.31. приведен пример использования этого свойства.

Application.StatusBar = "Пожалуйста, подождите..."

Листинг 9.31. Вывод фразы в StatusBar

**9.5.14. System - системная информация**

[09-08-System.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/9/files/09_08.docm) - пример к п. 9.5.12.

Это свойство возвращает объект System, который дает доступ к различной системной информации (о разрешении экрана, типе процессора, объеме свободного места на жестких дисках и т.д.)

Например, свойство FreeDiskSpace объекта System позволяет узнать размер свободного пространства на текущем жестком диске ([листинг 9.32](https://www.intuit.ru/studies/courses/494/350/lecture/8366?page=3#example.9.32).)

Dim var\_DiskSize As Variant

var\_DiskSize = Application.System.FreeDiskSpace

var\_DiskSize = Int(var\_DiskSize / (1024 ^ 2))

If var\_DiskSize >= 2047 Then \_

MsgBox ("На текущем HDD более 2 Гб свободного места") \_

Else MsgBox ("Размер свободного места на текущем диске " + \_

"равен " + Str(var\_DiskSize) + " Мб")

Листинг 9.32. Определение свободного дискового пространства

В этом примере мы учитываем тот факт, что при определении свободного места свойство FreeDiskSpace, во-первых, возвращает значение в байтах, а во-вторых - максимальное значение, возвращаемое функцией, не превышает 2,147,483,647 байт, то есть 2048 Мб или 2 Гб. Для получения значения в мегабайтах мы делим полученное значение на 1024 во второй степени.

**9.5.15. Tasks - работа с запущенными приложениями**

[09-09-Tasks.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/9/files/09_09.docm) - пример к п. 9.5.15.

Это свойство возвращает коллекцию Tasks, которая дает доступ ко всем приложениям, запущенным в системе. Давайте рассмотрим пример работы с приложением Notepad.exe (Блокнот), которое представлено в системе процессом Блокнот. Напишем программу ([листинг 9.33](https://www.intuit.ru/studies/courses/494/350/lecture/8366?page=3#example.9.33).), которая проверяет, запущен ли Блокнот, если запущен - выводит сообщение "Блокнот уже запущен", активирует его окно. Если Блокнот не запущен - запускает его, сообщив об этом.

If Application.Tasks.Exists("Блокнот") = True Then

MsgBox ("Блокнот уже запущен")

With Tasks("Блокнот")

.Activate

.WindowState = wdWindowStateMaximize

End With

Else

MsgBox "Запускаем Блокнот..."

Shell ("Notepad.exe")

With Tasks("Блокнот")

.Activate

.WindowState = wdWindowStateMaximize

End With

End If

Листинг 9.33. Работа с внешним приложением из MS Word

**9.5.16. Visible - скрытие окна MS Word**

Visible позволяет скрыть окно Microsoft Word. Если присвоить ему значение False - окно приложения будет скрыто, причем даже с панели задач Windows. Если значение True - окно снова отобразится.

**9.5.17. Windows - работа с окнами**

[09-10-Windows.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/9/files/09_10.docm) - пример к п. 9.5.17.

Это свойство возвращает коллекцию Windows, которая содержит окна всех открытых документов. Она позволяет управлять окнами. Например, код листинга 9.34. располагает окна на экране без перекрытия:

Application.Windows.Arrange (wdTiled)

Листинг 9.34. Располагаем окна на экране без перекрытия

Посмотрите на [листинг 9.35](https://www.intuit.ru/studies/courses/494/350/lecture/8366?page=3#example.9.35). Он иллюстрирует применение коллекции Windows:

For i = 1 To Application.Windows.Count

With Application.Windows.Item(i)

.Activate

.WindowState = wdWindowStateNormal

.Width = 400

.Height = 400

.Left = i \* 10 - 10

.Top = i \* 10 - 10

End With

Next i

Листинг 9.35. Работа с окнами документов

Здесь мы перебираем по очереди все окна (свойство Count коллекции Windows ), активируем каждое из них ( Activate ), после чего устанавливаем параметр WindowsState (Состояние окна) в значение wdWindowsStateNormal (иначе, если окно развернуто на весь экран, мы не сможем изменить его размеры и получим сообщение об ошибке). Далее, устанавливаем ширину и высоту окна по 400 пикселей и размещаем окна каскадом от левого верхнего угла экрана (координата 0,0), располагая каждое следующее окно правее и ниже предыдущего на 10 пикселей.

**9.6. События объекта Application**

[09-11-События.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/9/files/09_11.docm) - пример к п. 9.6.

Как и любые другие объекты, Application может реагировать на различные события. Чтобы события объекта отображались в редакторе, нужно воспользоваться особой методикой. Во-первых - вставьте в ваш *VBA*-проект *модуль* класса командой Insert - Class Module (Вставить - *Модуль* класса) или любым другим удобным для вас способом. Теперь разместить в модуле такое объявление ([листинг 9.36](https://www.intuit.ru/studies/courses/494/350/lecture/8366?page=3#example.9.36).).

Public WithEvents obj\_Application As Word.Application

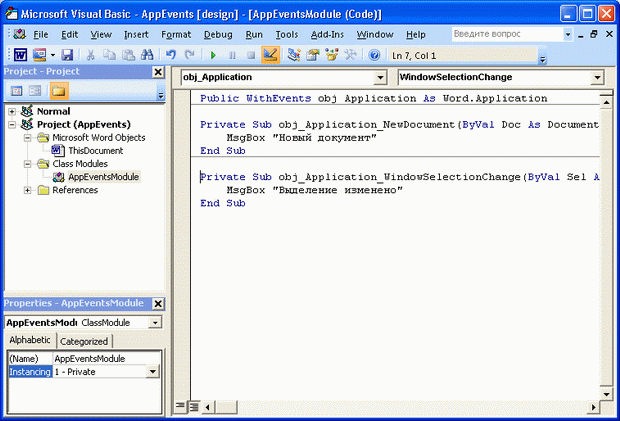
Листинг 9.36. Объявляем новую объектную переменную

После этого в списке объектов модуля появится новый *объект* obj\_Application, который будет представлять собой *приложение* Microsoft *Word*. В списке событий доступны для выбора различные события, соответствующие этому объекту. Например, следующие:

* DocumentBeforeClose (Перед закрытием документа) - событие генерируется непосредственно перед закрытием одного из открытых документов. В этот обработчик можно вставить, например, команды *проверки документа*, команды сохранения копий документа и т.д.
* DocumentBeforePrint (Перед печатью документа)
* DocumentBeforeSave (Перед сохранением документа)
* DocumentChange (Изменение документа) - это событие гененируется при создании нового документа, открытии существующего или тогда, когда любой документ становится активным.
* DocumentOpen (Открытие документа)
* NewDocument (Создание нового документа)
* Quit (Выход из Microsoft Word)
* WindowActivate (Активация окна документа)
* WindowBeforeDoubleClick (Перед двойным щелчком) - происходит после двойного щелчка в области редактирования документа до выполнения стандартной процедуры обработчика двойного щелчка.
* WindowBeforeRightClick (Перед щелчком правой кнопкой мыши)
* WindowDeactivate (Деактивация окна)
* WindowSelectionChange (Изменение выделенной области)
* WindowSize (Размер окна) - происходит при изменении размеров или перетаскивании окна.

Если вы создадите *модуль* класса, объявите в нем *объект* приложения, создадите обработчики, они не будут работать без специальной подготовки. Дело в том, что прежде чем обработчики для Word.Application заработают, нужно связать переменную, представляющую *приложение*, с приложением. Это можно сделать из какого-нибудь другого модуля. Например, чуть ниже мы познакомимся с автоматически выполняемыми макросами - их можно использовать для подобных целей.

Создадим новый документ Microsoft *Word*, в редакторе *VBA* добавим к нему новый *модуль* класса, назовем его AppEventsModule. Разместим в нем сначала объявление *объектной переменной* так, как указано выше, а потом создадим пару обработчиков, например, для событий NewDocument и WindowSelectionChange, в которые добавим код вызова окна сообщения с соответствующими сообщениями, указывающими на выполнение данного обработчика. Ниже, на [рис. 9.2](https://www.intuit.ru/studies/courses/494/350/lecture/8366?page=3#image.9.2). вы можете видеть окно проекта после создания модуля и добавления в него соответствующего кода.

[](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/9/files/09_02.gif)

[увеличить изображение](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/9/files/09_02.gif)  
**Рис. 9.2.**Окно проекта после добавления модуля класса

Теперь перейдем из редактора в документ, переключимся в *режим конструктора*, создадим в документе новую кнопку, назовем ее cmd\_initObject подпишем как Инициализация, и напишем для нее такой код ([листинг 9.37](https://www.intuit.ru/studies/courses/494/350/lecture/8366?page=3#example.9.37).):

Dim obj\_Module As New AppEventsModule

Private Sub cmd\_InitObject\_Click()

Set obj\_Module.obj\_Application = Word.Application

End Sub

Листинг 9.37. Связываем переменную с приложением

Мы не случайно привели здесь полный код обработчика - дело в том, что *переменная* obj\_Module должна быть объявлена не внутри обработчика, а в разделе объявлений модуля. Давайте подробнее рассмотрим этот код.

Выше мы не работали с модулями класса, поэтому сейчас нас ждут некоторые неожиданности. Первая неожиданность возникает при объявлении переменной obj\_Module. Как вы помните, при создании модуля класса мы задали его имя - AppEventsModule. Это имя может быть произвольным, главное - оно должно соответствовать общим правилам именования объектов и переменных *VBA*. В модуле мы объявили переменную obj\_Application, которая должна представлять *приложение*. Итак, объявляя переменную obj\_Module мы задаем для нее *тип данных* AppEventsModule - то есть создаем на основе класса AppEventsModule новый *объект*. В обработчике события нажатия кнопки cmd\_InitObject мы связываем переменную объекта obj\_Application (если следовать объектно-ориентированной терминологии, правильнее будет назвать ее свойством) с приложением Microsoft *Word*.

После того, как код обработчика для кнопки создан, выйдите из редактора *VBA* в документ, выключите *режим конструктора*, нажмите на кнопку и попытайтесь выполнить одно из действий, для которых в модуле AppEventsModule созданы обработчики. Если все сделано верно - вы увидите соответствующие этим обработчикам окна сообщений.

**9.7. Выводы**

В этой лекции мы обсудили *методы*, *свойства* и *события объекта* Application приложения Microsoft *Word*. На очереди - другие объекты. В частности, в следующей лекции мы поговорим о работе с документами.

Лекция 10. Работа с документами

**10.1. Коллекция Documents - открытые документы**

*Коллекция* Documents включает в себя открытые документы с их полезными методами и свойствами. Наряду с *объектами* Document, которые входят в *коллекцию*, здесь мы подробнее рассмотрим уже известный вам *объект* ActiveDocument,

ActiveDocument имеет те же свойства и методы, что и любой Document - *объект*, однако, часто *объектом* ActiveDocument удобнее пользоваться.

Так же обратите внимание на *объект* ThisDocument. Он представляет собой тот документ, из которого запущен *макрос*.

Для начала давайте рассмотрим свойства и методы *коллекции*, а потом займемся объектом Document.

**10.2. Методы коллекции Documents**

**10.2.1. Add - создаем новый документ**

Этот метод добавляет новый *объект* Document в *коллекцию* Documents - то есть создает новый документ. Его можно использовать так ([листинг 10.1](https://www.intuit.ru/studies/courses/494/350/lecture/8367?page=1#example.10.1).):

Documents.Add

Листинг 10.1. Создаем новый документ

При выполнении такого кода будет создан новый пустой документ на основе шаблона Normal. Чтобы ссылка на созданный документ была присвоена переменной, можете воспользоваться такой конструкцией ([листинг 10.2](https://www.intuit.ru/studies/courses/494/350/lecture/8367?page=1#example.10.2).):

Dim obj\_NewDoc As Word.Document

Set obj\_NewDoc = Application.Documents.Add

Листинг 10.2. Создаем новый документ и присваиваем ссылку на него объектной переменной

Здесь мы сначала объявили переменную соответствующего типа, а потом присвоили ей ссылку на новый документ.

Метод Add может принимать несколько необязательных параметров. Полный вызов метода выглядит так:

Documents.Add (Template, NewTemplate, DocumentType, Visible)

Template (Шаблон) - отвечает за шаблон, который будет использован при создании файла. По умолчанию это Normal.dotm.

NewTemplate (Новый шаблон) - если этот параметр установлен в True - файл будет обрабатываться как шаблон. По умолчанию установлен в False.

DocumentType (Тип документа) - очень полезный параметр, который позволяет создавать документы различных типов. На тип документа указывает одна из констант, в частности, по умолчанию это wdNewBlankDocument - то есть пустой документ. Так же этот параметр может принимать значения wdNewEmailMessage (Почтовое сообщение), wdNewFrameset (Документ с рамками - такие используются при создании WEB-страниц), wdNewWebPage (WEB-страница).

Visible (Видимость) - по умолчанию установлен в True - то есть создаваемый документ видим для пользователя. Если вы хотите, сначала сформировать новый документ, а уже потом показать его пользователю - установите это свойство в False, а после окончания формирования документа - воспользуйтесь свойством Visible созданного документа, чтобы отобразить его.

**10.2.2. Open - открываем документы**

Метод Open используется для открытия существующих файлов, возвращая объект типа Document. Файл, имя которого задано при вызове метода, открывается и добавляется в *коллекцию* Documents. Метод может принимать множество параметров, однако основной - это FileName, который и задает имя открываемого файла.

Например, для открытия файла 1.docm, расположенного в корневом каталоги диска С, можно воспользоваться кодом из [листинга 10.3](https://www.intuit.ru/studies/courses/494/350/lecture/8367?page=1#example.10.3).

Documents.Open FileName:="C:\1.docm"

Листинг 10.3. Открываем существующий документ

**10.2.3. Close - закрываем все документы**

Метод Close используется для закрытия всех открытых документов. Применяют его достаточно редко - удобнее закрывать каждый конкретный документ по отдельности. При вызове метода без параметров система задаст вопрос о сохранении документов, которые были изменены ([листинг 10.4](https://www.intuit.ru/studies/courses/494/350/lecture/8367?page=1#example.10.4).).

Documents.Close

Листинг 10.4. Закрываем все документы

Полный вызов метода выглядит так:

Documents.Close(SaveChanges, OriginalFormat, RouteDocument)

Среди параметров для нас наибольший интерес представляют первых два.

SaveChanges (Сохранить изменения) отвечает за сохранение изменений. По умолчанию он установлен в wdPromptToSaveChanges (Вывести запрос о сохранении), так же он может быть установлен в wdDoNotSaveChanges (Не сохранять изменения) и wdSaveChanges (Сохранить изменения). Как видите, с помощью параметра wdSaveChanges можно организовать удобное и быстрое закрытие множества документов с сохранением изменений

OriginalFormat (Формат) - отвечает за формат сохраняемого документа. Может принимать различные значения: wdOriginalDocumentFormat (Оригинальный формат документа), wdPromptUser (Запросить формат у пользователя), wdWordDocument (Документ Word).

Вот как выглядит вызов метода Close, который сохраняет все открытые документы и закрывает их ([листинг 10.5](https://www.intuit.ru/studies/courses/494/350/lecture/8367?page=1#example.10.5).)

Documents.Close (wdSaveChanges)

Листинг 10.5. Закрываем все документы, предварительно сохранив

Учтите - если какие-то из документов сохраняются в первый раз - вы получите запрос об имени документа.

**10.2.4. Item - обращаемся к отдельным документам**

[10-01-Item.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/10/files/10_01.docm) - пример к п. 10.2.4.

Метод Item позволяет обращаться к открытым документам (то есть - к объектам Document ), используя их индексы или имена. Нумерация документов начинается с единицы. Чтобы узнать общее количество открытых документов, можно воспользоваться свойством Documents.Count.

Например, с помощью нижеприведенного кода из [листинга 10.6](https://www.intuit.ru/studies/courses/494/350/lecture/8367?page=1#example.10.6) мы поочередно выводим имена открытых документов с указанием номера документа в *коллекции*:

For i = 1 To Documents.Count

MsgBox (Documents.Item(i).Name & \_

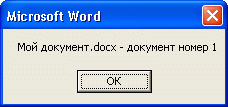
" - документ номер " & i)

Next i

Листинг 10.6. Выводим имена и номера документов

Очевидно, что свойство Count *коллекции* Documents позволяет узнать количество открытых документов, а свойство Name объекта Item(i) содержит имя документа с индексом i.

На [рис. 10.1](https://www.intuit.ru/studies/courses/494/350/lecture/8367?page=1#image.10.1) вы можете видеть одно из окон с сообщением об имени и номере документа.



**Рис. 10.1.**Окно сообщения об имени и номере документа

К документам в *коллекции* Documents можно обращаться и по именам. Также, при обращении к документу необязательно напрямую указывать метод Item - в [листинге 10.7](https://www.intuit.ru/studies/courses/494/350/lecture/8367?page=1#example.10.7) вы можете видеть различные варианты обращения к документу.

MsgBox Documents("Мой документ.docx").Name

MsgBox Documents.Item("Мой документ.docx").Name

MsgBox Documents(1).Name

Листинг 10.7. Выводим имена и номера документов

**10.2.5. Save - сохранение всех документов**

Метод Save сохраняет все документы *коллекции*. Если документ сохраняется впервые - будет выведено окно с запросом имени файла. В [листинге 10.8](https://www.intuit.ru/studies/courses/494/350/lecture/8367?page=1#example.10.8) приведен пример вызова метода.

Documents.Save

Листинг 10.8. Сохраним все документы в коллекции

**10.3. Свойства коллекции Documents**

Среди свойств *коллекции* Documents нас будет интересовать лишь одно - Count. Это свойство позволяет узнать количество элементов в *коллекции* Documents, то есть - количество открытых документов. В [листинге 10.9](https://www.intuit.ru/studies/courses/494/350/lecture/8367?page=1#example.10.9) мы выводим в окне сообщения количество открытых документов

MsgBox ("Количество открытых документов: " & \_

Documents.Count)

Листинг 10.9. Сохраним все открытые документы

**10.4. Объект Document**

*Объект* Document - это открытый документ. У него есть множество свойств и методов, которые активно используются при программной работе с документами. Здесь мы рассмотрим важнейшие из них.

**10.5. Методы объекта Document**

**10.5.1. Activate - сделаем документ активным**

Позволяет сделать документ активным. Например, это может понадобиться для редактирования документа.

Например, код из [листинга 10.10](https://www.intuit.ru/studies/courses/494/350/lecture/8367?page=2#example.10.10) активирует первый документ среди открытых документов.

Dim obj\_Doc As Document

Set obj\_Doc = Documents.Item(1)

obj\_Doc.Activate

Листинг 10.10. Активируем первый из открытых документов

**10.5.2. Close - закрыть документ**

Метод Close закрывает документ. Например, код из [листинга 10.11](https://www.intuit.ru/studies/courses/494/350/lecture/8367?page=2#example.10.11) позволяет закрыть документ, из которого он запущен.

ThisDocument.Close

Листинг 10.11. Закрываем документ, из которого запущен макрос

Метод можно вызывать с параметрами. В частности, полезен параметр SaveChanges, предназначенный для настройки автоматического сохранения документа. Если присвоить ему значение wdSaveChanges - документ будет сохранен при закрытии. Например, код из [листинга 10.12](https://www.intuit.ru/studies/courses/494/350/lecture/8367?page=2#example.10.12) по очереди закрывает каждый из открытых документов.

Dim obj\_Doc As Document

For Each obj\_Doc In Documents

obj\_Doc.Close SaveChanges:=wdSaveChanges

Next

Листинг 10.12. Закрываем каждый из открытых документов

При закрытии документов, которые были открыты и изменены, система выведет запрос о сохранении изменений.

**10.5.3. ComputeStatistics - статистика документа**

[10-02-Статистика документа.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/10/files/10_02.docm) - пример к п. 10.5.3

Полезный метод, позволяющий получать различные статистические сведения о документе. Полный вызов выглядит так:

ComputeStatistics(*Statistic*, IncludeFootnotesAndEndnotes)

Параметр *Statistic* позволяет задавать тип получаемой статистики. Он может принимать значения типа wdStatistic - константы, описание которых приведено ниже:

* wdStatisticCharacters -количество символов без пробелов
* wdStatisticCharactersWithSliaces - количество символов, включая пробелы
* wdStatisticLines - количество строк
* wdStatisticPages - количество страниц
* wdStatisticParagraphs - количество абзацев
* wdStatisticWords - количество слов

Параметр IncludeFootnotesAndEndnotes позволяет указать, учитывать ли при подсчете статистики сноски. Установленный в значение True, включает сноски в подсчет статистики, в False - исключает.

Чтобы подсчитать и вывести количество символов в документе без учета пробелов и с учетом сносок, нам понадобится код, приведенный в [листинге 10.13](https://www.intuit.ru/studies/courses/494/350/lecture/8367?page=2#example.10.13).

Dim num\_Stat

num\_Stat = ActiveDocument.ComputeStatistics \_

(wdStatisticCharacters, True)

MsgBox ("В документе " + Str(num\_Stat) + \_

" символов без пробелов, включая сноски")

Листинг 10.13. Подсчет статистики документа

**10.5.4. FitToPages - уменьшить размер текста на одну страницу**

Этот метод уменьшает размер документа на одну страницу - его удобно использовать, чтобы избавиться от страницы документа, на которой есть лишь несколько строчек.

**10.5.5. GoTo - перемещение по документу**

[10-03-GoTo.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/10/files/10_03.docm) - пример к п. 10.5.5.

Метод GoTo позволяет перемещаться по документу, "передвигать" позицию редактирования. Метод возвращает объект Range, который представляет собой место документа, на которое должен быть осуществлен переход. Его полный вызов выглядит так:

GoTo(What, Which, Count, Name)

What - тип части документа, на которую осуществляется переход. Например, среди них есть такие элементы:

* wdGoToBookmark - закладка (пожалуй, наиболее полезный элемент)
* wdGoToHeading - заголовок
* wdGoToLine - строка
* wdGoToObject - объект
* wdGoToPage - страница

Which - направление перемещения.

* wdGoToAbsolute - перемещение в абсолютных единицах измерения
* wdGoToFirst - перемещение к первому элементу
* wdGoToLast -перемещение к последнему элементу
* wdGoToNext - переход на следующий элемент
* wdGoToPrevious - переход на предыдущий элемент
* wdGoToRelative - перемещение относительно текущего элемента

Count - номер элемента в документе. Этот параметр часто опускают, так как переход осуществляется с использованием других установок, в частности - параметра Which.

Name - имя элемента, на который осуществляется переход. Например, имя может быть у закладки.

С помощью [листинга 10.14](https://www.intuit.ru/studies/courses/494/350/lecture/8367?page=2#example.10.14). мы можем перейти к первому заголовку в файле и добавить в его начало слово "Глава"

Dim obj\_Range As Range

Set obj\_Range = Application.Documents. \_

Item(1).GoTo(wdGoToHeading, wdGoToFirst)

obj\_Range.Text = "Глава " + obj\_Range.Text

Листинг 10.14. Переходы в пределах документа

**10.5.6. Range - область документа**

Возвращает объект Range, заданный позицией начального и конечного символа. Подробнее о нем мы поговорим в соответствующей лекции.

**10.5.7. Redo, Undo - отмена изменений**

Эти методы позволяют отменить последнее изменение ( *Redo* ) и вернуть отмененное изменение ( Undo ).

**10.5.8. RemoveDocumentInformation - удаляем персональную информацию из документа**

Позволяет удалить из документа различную персональную информацию. Например - данные о создателе документа и т.д. Метод может принимать параметр, который определяет, что именно следует удалять. На практике самым полезным является параметр wdRDIAll, который удаляет из документа все, что возможно .

**10.5.9. Save - сохранение файлов**

Сохраняет документ. Обычно применяют без параметров, хотя полезен может быть параметр NoPrompt. Установленный в True, предписывает MS Word сохранять файл без дополнительных вопросов пользователю.

Обычно для сохранения файлов с определенными параметрами используют метод SaveAs.

**10.5.10. SaveAs - тонкое управление сохранением**

Позволяет сохранить документ с новым именем или в новом формате. Обычно именно этот метод применяют и для первого сохранения новых документов, так как он позволяет очень тонко управлять параметрами сохранения. Всего этот метод принимает 16 параметров. Наиболее полезны среди них следующие:

FileName - задает имя файла

FileFormat - формат файла. Здесь могут быть использованы значения из перечисления wdSaveFormat. В частности, для сохранения документа в формате по умолчанию (.docx) надо использовать wdFormatDocumentDefault. Для сохранения документа в формате Word 97 - wdFormatDocument97, для получения HTML-файла - wdFormatHTML.

**10.5.11. Select - выделяем содержимое документа**

Выделяет все содержимое документа. После этого с документом можно работать, используя средства объекта Selection.

**10.5.12. UndoClear - очистить историю изменений**

Очищает историю изменений документа. После выполнения этого метода пользователь не сможет отменить изменения документа.

**10.6. Свойства объекта Document**

**10.6.1. Application - приложение документа**

Это свойство позволяет получить объект приложения документа. Вы можете подумать, что это - ошибка, что, наоборот, объект документа содержится в объекте приложения. Однако, ошибки нет. Это свойство позволяет обращаться к приложению тогда, когда у нас нет ссылки на него, а есть лишь ссылка на документ. Например, код в [листинге 10.15](https://www.intuit.ru/studies/courses/494/350/lecture/8367?page=3#example.10.15) позволяет скрыть окно Microsoft Word вместе с окном документа, используя свойство Visible объекта Application.

ThisDocument.Application.Visible = False

Листинг 10.15. Скрытие окна приложения

**10.6.2. AttachedTemplate - присоединенный шаблон**

Свойство AttachedTemplate позволяет узнать, какой шаблон присоединен к документу, а так же - изменить этот шаблон на другой. Это очень полезное свойство - ведь шаблоны могут содержать множество нужных макросов, стилей и т.д., а с помощью AttachedTemplate можно удобно управлять ими.

Например, такой [листинг 10.16](https://www.intuit.ru/studies/courses/494/350/lecture/8367?page=3#example.10.16). позволяет вывести имя присоединенного шаблона и путь к нему:

Dim obj\_Templ As Template

Set obj\_Templ = ActiveDocument.AttachedTemplate

MsgBox ("Имя шаблона: " & obj\_Templ.Name & \_

", путь к шаблону: " & obj\_Templ.Path)

Листинг 10.16. Имя присоединенного шаблона и путь к нему

Здесь мы сначала объявили новую переменную типа Template, потом присвоили ей ссылку на шаблон, присоединенный к активному документу, а потом воспользовались свойствами Name и Path шаблона, чтобы вывести информацию о нем.

[Листинг 10.17](https://www.intuit.ru/studies/courses/494/350/lecture/8367?page=3#example.10.17) позволяет присоединить к документу шаблон MyTemplate.dotm, расположенный на диске C.

ActiveDocument.AttachedTemplate = "C:\MyTemplate.dotm"

Листинг 10.17. Присоединяем шаблон к документу

**10.6.3. Bookmarks - закладки**

Возвращает коллекцию *Bookmarks*, которая содержит все закладки в документе. Закладки позволяют удобно организовать навигацию по документу, особенно удобно с их помощью автоматизировать создание новых документов на основе шаблонов. Подробнее о закладках мы поговорим ниже.

**10.6.4. Characters, Words, Sentences, Paragraphs и другие**

[10-04-Символы.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/10/files/10_04.docm) - пример к п. 10.6.4.

Все эти свойства возвращают объекты, предназначенные для работы с различными частями документа.

* Characters (Символы) - возвращает одноименную коллекцию, каждый объект которой представляет собой отдельный символ в документе. Каждый объект коллекции Characters представляет собой диапазон ( Range ), содержащий один символ
* Words (Слова) - то же самое, но уже для слов.
* *Sentences* (Предложения) - то же самое для предложений.
* Paragraphs (Абзацы) - коллекция абзацев документа.
* Помимо этих коллекций документ может содержать и другие подобные. В частности, следующие:
* *Hyperlinks* - гиперссылки. При добавлении гиперссылки обратите внимание на то, что метод Add использует объект привязки гиперссылки - Anchor (Якорь) - им может быть, например, графический или текстовый элемент.
* Indexes - индексы.
* InlineShapes, Shapes - фигуры, графические элементы
* Lists - списки.
* OMaths - формулы.
* Sections - разделы.
* Tables - таблицы.

Работа со всеми этими коллекциями схожа. Давайте рассмотрим ее на примере коллекции Characters, которую, как уже было сказано, можно использовать для работы с отдельными символами документа. Работу с некоторыми другими коллекциями мы рассмотрим ниже.

[Листинг 10.18](https://www.intuit.ru/studies/courses/494/350/lecture/8367?page=3#example.10.18) позволяет вывести в окне сообщения информацию о количестве символов в документе (включая пробелы).

MsgBox ("Количество символов: " & \_

ActiveDocument.Characters.Count)

Листинг 10.18. Выводим количество символов в документе

С помощью метода Item коллекции, мы можем получить доступ к каждому символу документа. Например, с помощью этого метода можно проводить различные статистические подсчеты. [листинг 10.19](https://www.intuit.ru/studies/courses/494/350/lecture/8367?page=3#example.10.19). позволяет узнать, сколько раз в документе использована буква "ф" в верхнем и нижнем регистре.

Dim var\_Fcount

var\_Fcount = 0

For i = 1 To ActiveDocument.Characters.Count

If ActiveDocument.Characters.Item(i) = "ф" Or \_

ActiveDocument.Characters.Item(i) = "Ф" Then \_

var\_Fcount = var\_Fcount + 1

Next i

MsgBox ("Количество букв 'Ф' в документе: " & \_

var\_Fcount)

Листинг 10.19. Подсчитываем количество вхождений буквы "ф"

[Листинг 10.20](https://www.intuit.ru/studies/courses/494/350/lecture/8367?page=3#example.10.20) позволяет заменить букву "ё" в верхнем и нижнем регистре на "е".

Dim obj\_D As Document

Set obj\_D = ActiveDocument

For i = 1 To obj\_D.Characters.Count

If obj\_D.Characters.Item(i) = "ё" \_

Then obj\_D.Characters.Item(i) = "е"

If obj\_D.Characters.Item(i) = "Ё" \_

Then obj\_D.Characters.Item(i) = "Е"

Next i

Листинг 10.20. Замена буквы е на е

В [листинге 10.21](https://www.intuit.ru/studies/courses/494/350/lecture/8367?page=3#example.10.21) вы можете найти программу, которая "переворачивает" весь текст в документе - меняет последний символ с первым, второй - с предпоследним и так далее.

Dim obj\_D As Document

Dim str\_Chr As String

Dim var\_Count As Variant

Set obj\_D = Documents.Add

obj\_D.Range.Text = "Привет!"

var\_Count = obj\_D.Characters.Count

For i = 1 To var\_Count \ 2

str\_Chr = obj\_D.Characters.Item(i)

obj\_D.Characters.Item(i) = \_

obj\_D.Characters.Item(var\_Count - i + 1)

obj\_D.Characters.Item(var\_Count - i + 1) = \_

str\_Chr

Next i

Листинг 10.21. Переворачивание текста в документе

Здесь мы сначала заводим переменную для хранения ссылки на активный документ, далее - строковую переменную для временного хранения одного символа и переменную типа Variant, в которой будем хранить количество символов в документе.

Далее создаем новый документ и добавляем в него текст "Привет" с помощью свойства Text объекта Range. Именно этот текст мы и будем переворачивать.

Итак, мы используем цикл длительностью, равной количеству символов в документе, деленному на 2, причем обратите внимание на оператор деления. Это - целочисленное деление, то есть для четного числа символов в обмене будут участвовать все символы, а для нечетного - один, "центральный" символ останется на своем месте.

В цикле мы сначала присваиваем i -й символ переменной, потом присваиваем i -му символу противоположный ему символ, а в конце - противоположному символу - символ, хранящийся в переменной. Обратите внимание на алгоритм расчета противоположного символа. Var\_Count - i + 1 дает нам последний символ при i = 1, то есть - для первого символа и т.д. В этом несложно убедиться путем расчетов. Например, если в тексте 10 символов, то для обмена с первым символом ( i = 1) мы получим последний символ (10-1+1=10), для второго ( i =2) - девятый (10-2+1=9) и т.д.

**10.6.5. Content - "чистый" текст документа**

[10-05-Content.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/10/files/10_05.docm) - пример к п. 10.6.5.

Свойство документа Content возвращает объект Range - текст документа без сносок, колонтитулов и т.д.

В [листинге 10.22](https://www.intuit.ru/studies/courses/494/350/lecture/8367?page=3#example.10.22) вы можете найти пример использования свойства Content для настройки шрифта документа.

Dim obj\_C As Range

Set obj\_C = ActiveDocument.Content

With obj\_C.Font

.Name = "Arial"

.Size = 12

.Spacing = 1.5

End With

Листинг 10.22. Настраиваем шрифт текста документа

В этом коде мы объявили *объектную переменную* типа Range, связали с ней объект Range, возвращаемый свойством Content активного документа, после чего, воспользовавшись свойством Font объекта Range, установили гарнитуру ( Name ) шрифта Arial, размер ( Size ) - 12 пунктов и расстояние между символами в 1.5 пункта.

Чтобы записать текст документа в строковую переменную, можно воспользоваться кодом [листинга 10.23](https://www.intuit.ru/studies/courses/494/350/lecture/8367?page=3#example.10.23).

Dim str\_Text As String

str\_Text = ActiveDocument.Content.Text

Листинг 10.23. Записываем текст документа в переменную

Cвойство Text объекта Range возвращает текст объекта. Записав текст в переменную, мы сможем работать с ней средствами *VBA*.

**10.6.6. Endnotes, Footnotes - сноски**

[10-06-Сноски.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/10/files/10_06.docm) - пример к п. 10.6.6.

Эти свойства возвращают, соответственно, коллекцию *Endnotes* и *Footnotes*. В коллекции *Endnotes* находятся объекты *Endnote* - то есть концевые ссылки - ссылки, которые расположены в конце документа. В коллекции *Footnotes* расположены объекты *Footnote* - обычные ссылки, которые расположены в конце страницы.

Давайте рассмотрим работу с этими объектами на примере *Footnotes*. Работа с *Endnotes* аналогична работе с *Footnotes*.

Добавить в документ новую ссылку можно с помощью метода Add коллекции *Footnotes*.

Полный вызов метода Add выглядит так:

* ActiveDocument.*Footnotes*.Add(Range, Reference, Text)
* Range - область текста, для которой вставляется сноска.
* Reference - определяет символ сноски. По умолчанию используется автоматическая нумерация сносок.
* Text - текст сноски.

В [листинге 10.24](https://www.intuit.ru/studies/courses/494/350/lecture/8367?page=3#example.10.24). вы можете видеть код, добавляющий сноску для выделенного фрагмента текста (как правило, выделяют слово):

ActiveDocument.Footnotes.Add \_

Range:=Selection.Range, \_

Text:="Новая сноска!"

Листинг 10.24. Добавление сноски к выделенному фрагменту документа

Свойство Count коллекции *Footnotes* позволяет узнать количество ссылок в документе.

Чтобы обращаться к отдельным сноскам (объектам *Footnote* ), можно пользоваться либо их индексами в коллекции *Footnotes*, либо - оператором For Each - Next.

Свойство Range каждого объекта *Footnote* возвращает ссылку на объект Range, представляющий собой диапазон, который сноска занимает в документе. С помощью методов объекта можно настраивать различные параметры сноски.

Свойство Reference возвращает объект типа Range, который представляет собой область, занимаемую значком-ссылкой на сноску в *теле документа*.

В [листинге 10.25](https://www.intuit.ru/studies/courses/494/350/lecture/8367?page=3#example.10.25) приведен пример форматирования сносок и ссылок на них.

Dim obj\_D As Document

Dim obj\_Footnote As Footnote

Set obj\_D = ActiveDocument

'Форматируем сноски

For Each obj\_Footnote In obj\_D.Footnotes

obj\_Footnote.Range.Font.Size = 11

Next

'Форматируем знаки-ссылки на сноски

For i = 1 To obj\_D.Footnotes.Count

obj\_D.Footnotes.Item(i).Reference \_

.Font.Color = wdColorRed

Next i

Листинг 10.25. 10.25. Форматирование сносок и ссылок на них

Сначала мы, с помощью конструкции For-Each перебираем все сноски и устанавливаем размер шрифта для каждой из них равным 11 пунктов.

Далее, с помощью цикла For-Next мы обходим все ссылки на сноски в тексте и делаем цвет их шрифта красным.

Чтобы удалить сноску, можно воспользоваться методом Delete объекта *Footnote*.

Узнать номер сноски в коллекции *Footnotes* можно, используя свойство Index. Например, код [листинга 10.26](https://www.intuit.ru/studies/courses/494/350/lecture/8367?page=3#example.10.26) позволяет вывести поочередно все сноски с их номерами в окне сообщения.

Dim obj\_D As Document

Dim obj\_Footnote As Footnote

Set obj\_D = ActiveDocument

'Выводим информацию о сносках

For Each obj\_Footnote In obj\_D.Footnotes

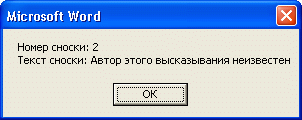
MsgBox ("Номер сноски: " & obj\_Footnote.Index & \_

vbCrLf & "Текст сноски: " & obj\_Footnote.Range.Text)

Next

Листинг 10.26. 10.26. Выводим все сноски и их номера

Константа vbCrLf заменяет собой знак перевода строки, в итоге при выполнении этого кода для каждой сноски будет выведено такое окно ([рис. 10.2](https://www.intuit.ru/studies/courses/494/350/lecture/8367?page=3#image.10.2).).



**Рис. 10.2.**Информация о сносках

**10.6.7. FullName, Name, Path - пути и имена**

Свойство FullName возвращает строку, содержащую полное имя документа - то есть полный путь к документу и его имя.

Листинг 10.27. выводит полное имя документа в окне сообщения.

MsgBox (ActiveDocument.FullName)

Листинг 10.27. Выводим полное имя документа

Свойство Name возвращает имя документа без пути, а Path - путь без имени.

**10.6.8. Saved - сохранен или нет**

Возвращает True если документ не изменялся после последнего сохранения.

**10.6.9. Tables - работа с таблицами**

[10-07-Таблицы.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/10/files/10_07.docm) - пример к п. 10.6.9.

Свойство возвращает коллекцию Tables (Таблицы), которая содержит объекты Table - таблицы, которые имеются в документе. Работа с таблицами похожа на работу с другими объектами документов.

Например, чтобы добавить в документ новую таблицу, нужно воспользоваться методом Add коллекции Tables. Рассмотрим пример ([листинг 10.28](https://www.intuit.ru/studies/courses/494/350/lecture/8367?page=4#example.10.28).). Добавим в позицию курсора таблицу, состоящую из 3-х строк и 4-х столбцов.

Dim obj\_Table As Table

Set obj\_Table = \_

ActiveDocument.Tables.Add(Range:=Selection.Range, \_

NumRows:=3, NumColumns:=4)

With obj\_Table

.Borders.OutsideLineStyle = wdLineStyleTriple

.Borders.InsideLineStyle = wdLineStyleSingle

End With

Листинг 10.28. Добавляем в документ новую таблицу

Здесь мы сначала объявляем *объектную переменную* типа Table - для хранения ссылки на таблицу. Далее - при помощи метода Add коллекции Tables добавляем в документ новую таблицу, состоящую из 3-х строк и 4-х столбцов. Обратите внимание на параметр Range метода Add. Он задает позицию в документе, куда будет вставлена новая таблица. Здесь мы приравняли ему такое значение: Selection.Range. Ниже мы будем говорить и об объекте Selection, и о Range, однако отметим, что выражение Selection.Range можно передать обычным языком как "Текущее выделение". Значит, таблица будет вставлена в выделенную область документа, либо, если область документа явно не выделена - туда, где в момент запуска макроса установлен курсор.

Добавленная таблица не очень хорошо заметна в документе - она состоит из невидимых линий. Поэтому мы настраиваем отображение линий таблицы с помощью блока With - End With для переменной obj\_Table. Воспользуемся коллекцией Borders объекта Table - она позволяет работать с границами таблицы. Параметр OutsideLineStyle позволяет задавать тип линий внешней границы таблицы. Здесь мы установили тип линии wdLineStyleTriple - это тройная линия. Аналогично, с помощью параметра InsideLineStyle мы сделали внутренние линии таблицы одинарными ( wdLineStyleSingle ).

**10.6.10. Omaths - математические формулы**

Свойство OMaths возвращает коллекцию OMaths, которая содержит математические формулы, хранящиеся в документе. Как и в случае с другими подобными коллекциями, OMaths имеет множество полезных свойств и методов. Давайте рассмотрим пример добавления в документ новой математической формулы ([листинг 10.29](https://www.intuit.ru/studies/courses/494/350/lecture/8367?page=4#example.10.29)). Этим кодом мы добавим новую формулу, преобразовав в нее выделенный фрагмент документа.

Selection.OMaths.Add Selection.Range

Selection.Range.OMaths(1).BuildUp

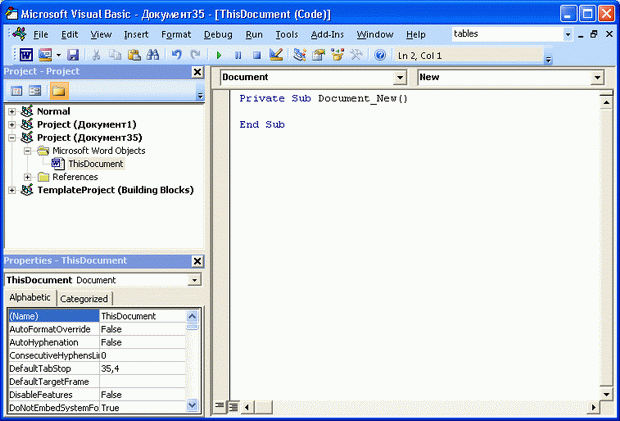
Листинг 10.29. Добавляем в документ новую формулу

Итак, для добавления нового объекта OMath в коллекцию OMath, мы используем метод Add. Главный параметр, который нужно передать методу Add - это объект типа Range, содержащий текст, который должен быть преобразован в формулу. В нашем случае это конструкция Selection.Range. Проще говоря, выделенный текст. Далее мы используем коллекцию OMaths для объекта Range текущего выделения. Она будет содержать все объекты OMath в выделении. Мы обращаемся к первому (и единственному) элементу коллекции и выполняем метод BuildUp - этот метод преобразует текст объекта в профессиональный математический вид.

**10.7. События объекта Document**

[10-08-Documents События.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/10/files/10_08.docm) - пример к п. 10.7.

Работать с событиями документа гораздо проще, чем с событиями приложения. Чтобы создать обработчик события для документа нужно войти в редактор *Visual* *Basic* for Applications, выбрать в окне Project Explorer *объект* ThisDocument - этот *объект* представляет собой документ, в котором хранится *VBA*-проект. Далее, откроется окно редактирования кода, в нем, в списке объектов, нужно выбрать Document - этот *объект* символизирует документ, которому принадлежит проект, а в окне выбора события - событие, для которого мы хотим создать обработчик. Вот, как ([рис. 10.3](https://www.intuit.ru/studies/courses/494/350/lecture/8367?page=4#image.10.3).) выглядит окно редактора после создания пустого обработчика события New для документа.

[](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/10/files/10_03.gif)

[увеличить изображение](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/10/files/10_03.gif)  
**Рис. 10.3.**Добавление обработчика события для документа (10\_03.bmp)

Событие New добавляется по умолчанию при выборе объекта Document в списке объектов редактора. Однако, это событие пригодно лишь для использования его в документах, которые планируется сохранить как шаблоны и использовать для автоматизации создания других документов. Ниже мы рассмотрим пример работы с шаблонами.

New - одно из самых полезных и часто используемых событий документа. Помимо него можно отметить события Open, которое выполняется при открытии документа, и Close - оно выполняется при закрытии. Эти события работают и с обычными документами, а не только с шаблонами, как New.

Open и Close используют для настройки проверок документа перед закрытием, для инициализации событий приложения ( Open ) и в других подобных случаях. Если вам нужно сделать что-то до начала редактирования документа или перед его закрытием - воспользуйтесь событиями Open и Close.

**10.8. Выводы**

В этой лекции мы рассмотрели особенности программной работы с документами MS *Word*. На очереди - работа с объектами, представляющими собой различные участки документов.

Лекция 11. Редактирование документов, автоматически выполняемые макросы

**11.1. Объект Selection**

Выше нам уже встречался и *объект* Selection, и *объект* Range. Здесь пришло время поговорить о них подробнее, обсудить особенности их использования.

Напомню, что *объект* Application.Selection представляет собой выделенный участок документа или, если ничего не выделено, позицию курсора. Именно этим объектом пользуется *макрорекордер* при записи макроса, выполняющего какие-либо *операции* с текстом.

У Selection есть два основных применения. Первое — *автоматическое создание документов*, не требующее правки *шаблонов* и других подобных операций. И второе — работа с фрагментом документа, предварительно выделенного пользователем.

Выделения можно создавать и программно, но эту методику практически никогда не используют на практике. Например, код [листинга 11.1](https://www.intuit.ru/studies/courses/494/350/lecture/8368?page=1#example.11.1) выделяет седьмое *слово* в документе и форматирует его курсивом:

ActiveDocument.Words.Item(7).Select

Selection.Font.Italic = True

Листинг 11.1. Выделяем седьмое слово и форматируем его

Предположим, мы используем такую конструкцию в *шаблоне* для автоматического форматирования текста. Если мы правильно рассчитаем номер слова, которое нужно отформатировать (или отредактировать, вставить в документ и т.д.) — то все будет работать правильно. Но стоит нам отредактировать *шаблон* — даже совсем немного, но так, что количество слов в нем изменится, наш *макрос* будет работать неправильно.

*Объект* Selection обычно используют без объявления *объектной переменной*. Например, для вставки в выделенную позицию документа текста "Привет" достаточно конструкции [листинга 11.2](https://www.intuit.ru/studies/courses/494/350/lecture/8368?page=1#example.11.2).

Selection.Text = "Привет"

Листинг 11.2. Добавляем текст в выделенную позицию документа

Минус использования Selection при автоматической обработке текста зависит от того, что он слишком чувствителен к действиям пользователя. Например, мы напишем программу, которая, имитируя выделение фрагментов текста с клавиатуры, обрабатывает текст. Если во время обработки текста *пользователь* вмешается, случайно кликнув мышью в документе, работа такой программы будет нарушена.

Рассмотрим основные свойства и методы *объекта* Selection.

**11.2. Методы объекта Selection**

**11.2.1. ClearFormatting — очистить форматирование**

Этот метод очищает форматирование выделенного участка документа.

**11.2.2. Copy, Cut, Paste, Delete**

Методы Copy и Cut, соответственно, копируют и вырезают выделенный фрагмент в буфер обмена, а метод Paste — вставляет материалы из буфера в область выделения.

Метод Delete удаляет выделенный фрагмент

**11.2.3. InsertAfter, InsertBefore — добавление текста**

Как следует из их названий, методы InsertAfter и InsertBefore позволяют вставить текст после или до выделения. Существуют и другие методы, название которых начинается с Insert. Они позволяют вставлять множество других объектов в выделенную область. Например, InsertBreak вставляет разрыв страницы, колонки или раздела, InsertCells вставляет в выделенную таблицу ячейки и т.д.

**11.2.4. TypeBackspace — удалить символ**

Удаляет один символ, предшествующий позиции курсора, имитируя нажатие клавиши Backspace на клавиатуре.

**11.2.5. TypeParagraph — перевод строки**

Имитирует нажатие клавиши Enter на клавиатуре — вставляет в текст знак перевода строки.

**11.2.6. TypeText — вывести текст в документ**

Вводит в документ текст. У метода есть лишь один параметр — Text. Он должен содержать тот текст, который должен быть выведен в документ. Ниже мы рассмотрим примеры, где используется этот метод.

**11.2.7. WholeStory — выделить весь документ**

Позволяет выделить весь документ. Пожалуй, это единственный удобный метод программного управления выделением.

**11.3. Свойства объекта Selection**

**11.3.1. Bookmarks и другие**

Это свойство возвращает коллекцию *Bookmarks*, которая содержит все закладки, имеющиеся в выделении. Существуют подобные свойства для символов ( Characters ), таблиц ( Tables ) и многих других коллекций объектов. Работа с ними ни чем не отличается от работы с аналогичными коллекциями, входящими в Document.

**11.3.2. End и Start**

Эти свойства позволяют узнать и модифицировать, соответственно, конечную ( End ) и начальную ( Start ) позиции выделения.

**11.3.3. Font — шрифт**

Это свойство позволяет узнавать и настраивать параметры шрифта выделенного фрагмента. Например, код, приведенный в [листинге 11.3](https://www.intuit.ru/studies/courses/494/350/lecture/8368?page=1#example.11.3) форматирует текст. Пояснения к примеру приведены в комментариях.

'Полужирный

Selection.Font.Bold = wdToggle

'Курсивный

Selection.Font.Italic = wdToggle

'Подчеркнутый

Selection.Font.Underline = wdUnderlineSingle

'Гарнитура Arial Narrow

Selection.Font.Name = "Arial Narrow"

'Размер шрифта - 14

Selection.Font.Size = 14

'Цвет шрифта - красный - из

'перечисления wdColorR

Selection.Font.Color = wdColorRed

Листинг 11.3. Форматируем текст

**11.3.4. Information — подробности о выделении**

Позволяет узнать подробности о выделении. Это свойство может принимать параметр типа wdInformation, который и определяет, что именно возвратит свойство.

**11.3.5. Range — переход от Selection к Range**

Это свойство позволяет получить *объект* Range, соответствующий выделению.

Например, этим свойством можно воспользоваться так, как показано в [листинге 11.4](https://www.intuit.ru/studies/courses/494/350/lecture/8368?page=1#example.11.4).

Dim obj\_Range As Range

Set obj\_Range = Selection.Range

Листинг 11.4. Получаем объект Range из Selection

**11.3.5. Text — текст выделения**

Это свойство позволяет получить текст, содержащийся в выделении, и изменить его.

**11.4. Пример работы с Selection**

[11-01-Selection.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/11/files/11_01.docm)— пример к п. 11.4.

Рассмотрим пример использования вышеперечисленных свойств и методов (а заодно – и некоторых других), решив небольшую задачу.

**11.4.1. Задание**

Создать новый документ, и добавить в него текст, который был выделен в текущем документе. Каждое слово этого текста надо напечатать отдельно, с новой строки. Каждое слово должно предваряться таким текстом: "Слово №x:", где x — номер слова в выделенном тексте. После вывода слов нужно подсчитать количество слов и количество букв в этих словах и вывести строку "Всего обработано "x" слов, состоящих из "xx" букв". Готовый документ сохранить на диске C:\, задав в качестве его имени такую структуру:

"Имя\_исходного\_документа"+ "текущая дата"+ "обработано".docm

После того, как слова будут выведены, нужно заключить выделенный участок исходного документа в скобки, отметить синим цветом шрифта и выделить первый символ текущего выделения.

**11.4.2. Решение**

Создадим новый макрос ( Copy\_To\_File ), воспользовавшись окном Макросы, которое можно открыть с вкладки Вид.

Добавим в макрос код из [листинга 11.5](https://www.intuit.ru/studies/courses/494/350/lecture/8368?page=1#example.11.5).

'Переменая для хранения ссылки на новый документ

Dim obj\_NewDoc As Document

'Для ссылки на исходный документ

Dim obj\_OurDoc As Document

'Динамический массив для хранения слов

Dim WordsArray() As String

'Переменная для хранения количества слов

Dim WordsCount

'Переменная для формирования строк,

'которые выводятся в новый документ

Dim OutString

'Переменная для хранения количества

'символов в обработанных словах

Dim LettersCount

'В этой переменной будем создавать

'имя файла

Dim NewDocName

'Запишем ссылку на активный документ

'Для того, чтобы вернуться в него

'после работы с новым документом

Set obj\_OurDoc = ActiveDocument

WordsCount = Selection.Words.Count

LettersCount = Selection.Characters.Count

ReDim WordsArray(WordsCount)

'Переносим слова из выделения

'в массив для удобства работы с ними

For i = 1 To WordsCount

WordsArray(i) = Selection.Words.Item(i)

Next i

'Создадим новый документ

Set obj\_NewDoc = Documents.Add

'И сделаем его активным

obj\_NewDoc.Activate

'Теперь объект Selection относится к активному

'документу и мы начинаем вводить в него данные

Selection.TypeText ("Список выделенных слов")

Selection.TypeParagraph

For i = 1 To WordsCount

OutString = "Слово №" + Str(i) + ": " + \_

WordsArray(i)

Selection.TypeText (OutString)

Selection.TypeParagraph

Next i

Selection.TypeText ("Всего обработано " + \_

Str(WordsCount) + \_

" слов(а), в которых содержится " + \_

Str(LettersCount) + " символа(ов)")

'Создаем имя нового документа

NewDocName = obj\_OurDoc.Name + " " + \_

Str(Date) + " Обработано.docx"

'Установим корневой каталог диска C:

'для сохранения нового файла по умолчанию

ChangeFileOpenDirectory "C:\"

'Используем метод SaveAs для

'автоматического сохранения

'документа

ActiveDocument.SaveAs \_

FileName:=NewDocName, \_

FileFormat:=wdFormatDocumentDefault

'Закрываем активный документ

ActiveDocument.Close

'Активируем документ, в котором мы выделяли

'слова

obj\_OurDoc.Activate

'Отмечаем обработанный участок

'скобками и выделением синим цветом

Selection.InsertAfter (") ")

Selection.InsertBefore (" (")

Selection.Font.Color = wdColorBlue

'Выделяем первый символ текущего выделения и

'выводим сообщение о том, что операция

'выполнена

Selection.Characters(1).Select

MsgBox ("Выполнено!")

Листинг 11.5. Решение задачи

Как видите, в этом примере мы используем исключительно *объект* Selection для работы с текстом документов. Обратите внимание на то, что мы нигде специально не храним информацию о выделенной области первого документа. Создав второй документ, мы просто делаем его активным и работаем с ним, используя его *объект* Selection. В это время выделение сохраняется в исходном документе. Однако, если в это время пользователь случайно или умышленно изменит выделение текущего документа — программа будет работать неправильно — она выделит не тот фрагмент, который был передан ей для обработки, а совсем другой.

**11.5. Объект Range**

*Объект* Range имеет некоторые преимущества перед *объектом* Selection. В частности, при работе с Range нет необходимости выделять текст или перемещать по тексту *курсор*.

В документе может быть сколько угодно *объектов* Range, в то время, как *объект* Selection может быть лишь один. *Объект* Range задается первым и последним символом.

Особенность работы с Range заключается в том, что этот *объект* может быть создан лишь программно. Получить Range для различных частей документа можно разными способами.

**11.5.1. Как получить объект Range**

Можно преобразовать в Range выделение и больше не зависеть от действий пользователя — с помощью свойства Range *объекта* Selection. В ["Основы VBA: встроенные и пользовательские функции"](https://www.intuit.ru/studies/courses/494/350/lecture/8361)мы рассматривали пример такого перехода.

Объект Range, в свою очередь, может быть преобразован в выделение — с помощью метода Select. Поэтому мы можем легко переходить от *объектов* Range к *объектам* Selection и наоборот.

Можно задать Range напрямую — указав диапазон. Например, следующий код позволяет получить диапазон активного документа от 10 до 1500-го символа ([листинг 11.6](https://www.intuit.ru/studies/courses/494/350/lecture/8368?page=2#example.11.6).)

Dim obj\_Range As Range

Set obj\_Range = \_

ActiveDocument.Range(10, 1500)

Листинг 11.6. Задаем объект Range напрямую

Минус этого метода очевиден: чтобы программно работать с определенным участком документа, нужно знать его "координаты" — даже если предварительно вычислить их, любое изменение документа вызовет необходимость пересчета.

Можно получить Range для любого объекта документа, обратившись к соответствующему свойству объекта типа Character, Word и т.д. Например, следующий код ([листинг 11.7](https://www.intuit.ru/studies/courses/494/350/lecture/8368?page=2#example.11.7).) записывает в *объектную переменную* ссылку на Range для 5-го слова активного документа.

Dim obj\_Range As Range

Set obj\_Range = \_

ActiveDocument.Words(5).Range

Листинг 11.7. Получаем объект Range из объекта Word (слово)

Очень часто используется разновидность этого метода, связанная со свойством Range объекта Bookmark — закладки. На практике это — один из самых удобных способов *автоматического создания документов* на основе *шаблонов*. Ниже мы подробнее поговорим об использовании закладок.

**11.5.2. Свойства и методы объекта Range**

Если вы ориентируетесь в методах и свойствах *объекта* Selection — это значит, что вы сможете работать и с Range. Между этими объектами очень много общего. Здесь мы отметим лишь уже упомянутый метод Select *объекта* Range — он позволяет выделить диапазон, соответствующий ему, а так же свойство Text, которое предназначено для модификации текста, который содержится в Range. Практику работы с Range мы рассмотрим в следующем разделе, когда будем говорить о закладках.

**11.6. Коллекция Bookmarks, шаблоны**

[11-02-Закладки и шаблоны.dotm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/11/files/11_02.dotm) — пример к п. 11.6.

Коллекция *Bookmarks* содержит закладки. Эта коллекция может использоваться совместно с объектами Document, Selection и Range. Причем, когда речь идет о коллекции Document.*Bookmarks*, мы можем работать с закладками в масштабах всего документа, коллекция Selection.*Bookmarks* содержит закладки в пределах выделения, а Range.*Bookmarks* — в пределах диапазона.

Закладки, добавленные в документы, хранятся там постоянно. Объекты *Bookmarks* имеют очень мало свойств и методов. Главная задача закладки — это точно указать *место* в документе, а все остальное — это уже забота либо *объектов* Selection, либо — Range. Для получения *объекта* Range из закладки достаточно воспользоваться ее свойством Range.

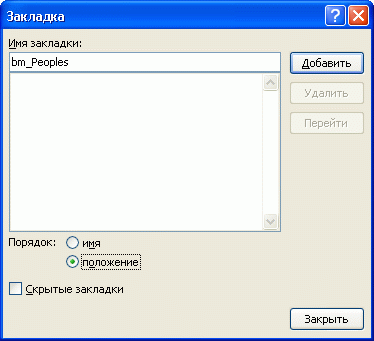
С помощью закладок очень удобно автоматизировать создание документа. Зная имя или позицию нужной закладки в коллекции, можно обращаться к ней, вставляя определенный текст. В частности, использование закладок для автоматизации создания документа может предусматривать следующую схему действий.

* Вручную создать *шаблон*, содержащий постоянные части и переменные части, выделяемые закладками.
* Создать новый документ, используя *шаблон*
* Заполнить новый документ данными, используя закладки

Давайте рассмотрим пример, который иллюстрирует вышеописанные шаги. Мы будем создавать *шаблон* приглашения с несколькими переменными частями.

**11.6.1. Закладки и шаблон**

Чтобы создать *шаблон*, создадим для начала новый документ. После этого наберем и отформатируем в документе текст приглашения: "Уважаем ый Александр Александрович! Приглашаем вас на собрание, которое состоится 17.05.08. Администрация" ([рис. 11.1](https://www.intuit.ru/studies/courses/494/350/lecture/8368?page=2#image.11.1).).

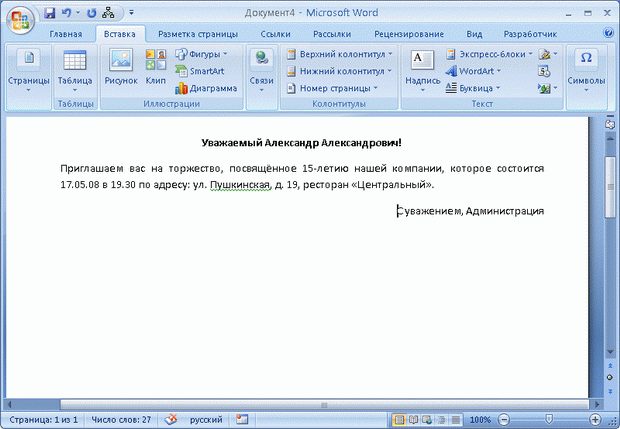


**Рис. 11.1.**Подготовка текста шаблона

В этом тексте выделены места, которые могут меняться. Давайте создадим таблицу, содержащую информацию об этих местах документа ([табл. 11.1](https://www.intuit.ru/studies/courses/494/350/lecture/8368?page=2#table.11.1).).

|  |  |  |
| --- | --- | --- |
| Таблица 11.1. Изменяемые части документа и закладки | | |
| **Изменяемый элемент** | **Описание** | **Имя и номер закладки** |
| ый | Окончание, определяющее лицо или лица, которым адресовано послание. Возможные варианты: "ый" - для мужчины, "ая" - для женщины, "ые" - для группы лиц | bm\_Peoples, №1 |
| Александр Александрович | Имя приглашенного | bm\_Name, №2 |
| 17.05.08 | Дата | bm\_Date, №3 |

Чтобы назначить элементу документа закладку, нужно выделить этот элемент (например, слово или часть слова), после чего перейти на вкладку Вставка, нажать там кнопку Связи и в появившемся меню выбрать пункт Закладки. Появится окно, в котором нужно задать имя закладки и нажать на кнопку Добавить для добавления закладки в документ ([рис. 11.2](https://www.intuit.ru/studies/courses/494/350/lecture/8368?page=2#image.11.2).).

[](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/11/files/11_02.gif)

[увеличить изображение](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/11/files/11_02.gif)  
**Рис. 11.2.**Добавление закладки в документ

После того, как закладки добавлены, оснастим наш *шаблон* средствами для автоматического редактирования новых документов, созданных на его основе. Текст программы мы поместим в обработчик события New *шаблона*.

В нашем случае это должен код ([листинг 11.8](https://www.intuit.ru/studies/courses/494/350/lecture/8368?page=2#example.11.8).), который запрашивает данные у пользователя. При необходимости можно видоизменить программу таким образом, чтобы она брала данные из специально подготовленного файла, из базы данных, можно создать форму для ввода всех значений по каждому из пользователей и, запустив ее, создать необходимое количество документов, используя *шаблон* и т.д.

Private Sub Document\_New()

Dim obj\_NewDoc As Document

Dim obj\_Range As Range

Set obj\_NewDoc = Application.Documents.Item(1)

Set obj\_Range = \_

obj\_NewDoc.Bookmarks("bm\_Peoples").Range

obj\_Range.Text = \_

InputBox("Введите окончание обращения: -ый, -ая, -ые")

Set obj\_Range = \_

obj\_NewDoc.Bookmarks("bm\_Name").Range

obj\_Range.Text = \_

InputBox("Введите имена приглашенных")

Set obj\_Range = \_

obj\_NewDoc.Bookmarks("bm\_Date").Range

obj\_Range.Text = \_

InputBox("Введите дату собрания")

End Sub

Листинг 11.8. Код для автоматического формирования нового документа на основе шаблона

Сначала мы объявляем пару *объектных переменных*. Одну — для документа, с которым будем работать, другую — для диапазона документа, который будем выбирать с помощью закладки и после этого редактировать.

В первую очередь получаем ссылку на созданный документ. Он будет первым в коллекции Application.Documents.

Далее — записываем в *объектную переменную* obj\_Range диапазон, которому соответствует закладка с именем bm\_Peoples — то есть та самая, которая должна хранить окончание обращения.

Следующий шаг — воспользоваться свойством Text *объекта* Range и предложить пользователю ввести окончание для обращения.

Далее процесс повторяется — ссылка на диапазон очередной закладки записывается в переменную, после чего этот диапазон модифицируется путем ввода данных.

Здесь мы рассмотрели лишь общую схему работы с закладками. На практике неудобно будет вводить большое количество информации в окна запросов. Поэтому данный пример можно развивать в нескольких направлениях.

1. Создать форму с текстовыми полями для ввода данных. После ввода достаточно нажать на кнопку, чтобы на основе *шаблона* был сформирован новый документ, который можно сохранить на диске.
2. Создать отдельный документ MS Word, в котором будут перечислены изменяемые части документа. Например, если рассмотреть обычное приглашение на какое-то мероприятие, в изменяемую часть приглашения входит лишь обращение к приглашаемым. Это значит, что в документ, создаваемый вручную, можно включить список имен и фамилий приглашенных с обращениями к ним, после чего создать *шаблон* и написать макрос, который автоматически подставит данные из списка в документы, созданные по *шаблону*. Этот метод удобен при работе с небольшими (до нескольких сотен) списками и может быть реализован стандартными средствами слияния документов MS Word.
3. Использовать в качестве источника информации базу данных или документ другого типа. Например, можно воспользоваться базой данных MS Access или документом MS Excel, содержащим список приглашенных. О работе с базами данных и документами MS Excel мы поговорим ниже.

**11.7. Автоматически запускаемые макросы**

[11-03-Автоматические макросы.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/11/files/11_03.docm) — пример к п. 11.7.

Работая с макросами Microsoft *Word*, следует знать о том, что существует несколько имен, зарезервированных для макросов, которые могут выполняться автоматически. Имена и события, при возникновении которых запускаются эти макросы, приведены в [табл. 11.2](https://www.intuit.ru/studies/courses/494/350/lecture/8368?page=2#table.11.2).

|  |  |
| --- | --- |
| Таблица 11.2. Автоматически выполняемые макросы | |
| **Имя макроса** | **Условия выполнения** |
| AutoExec | Старт Microsoft Word или загрузка глобального шаблона. Этот макрос будет работать лишь из Normal.dotm |
| AutoNew | Создание нового документа |
| AutoOpen | Открытие существующего документа |
| AutoClose | Закрытие документа |
| AutoExit | Выход из Microsoft Word или выгрузка глобального шаблона |

Давайте добавим к документу *макрос* AutoClose — он будет выполняться каждый раз при закрытии документа. Для этого с помощью панели Вид откроем окно Макрос, в *поле* ввода имени макроса введем AutoClose и нажмем на кнопку Создать. Появится окно редактора *Visual* *Basic*, содержащее новый *модуль* с макросом, имеющим имя AutoClose. Добавим в код макроса вызов окна сообщения, выводящего надпись "До свидания!". Ведь *макрос* AutoClose будет выполняться при выходе из документа, логично будет попрощаться с пользователем после того, как тот нажал на кнопку закрытия документа.

Вы могли заметить, что эти макросы очень похожи на события объекта Document — их можно использовать в различных целях, когда нужна предварительная подготовка документа после его создания или проведение каких-либо *работ* сразу перед закрытием. Будучи добавленными в Normal.dotm, такие макросы могут послужить целям автоматизации работы в MS Office.

**11.8. Выводы**

В этой лекции мы познакомились с приемами автоматического редактирования и создания документов MS *Word* с использованием *объектов* Range и Selection. Так же мы рассмотрели пример автоматического создания документа на основе *шаблона* и работу с автоматически запускаемыми макросами. В следующей лекции вы найдете несколько примеров программирования в MS *Word* на *VBA*.

Лекция 12. Практические примеры программирования для MS Word

**12.1. Групповая обработка файлов**

[12-01-Групповая обработка файлов.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/12/files/12_01.docm) - пример к п. 12.1.

**12.1.1. Условие**

Предположим, у вас есть несколько сотен файлов в формате .docx и . docm. Например, с ними вы работали дома. Очевидно, что дома у вас установлен Office 2007. И вот, приходит время перенести эти материалы на рабочий компьютер. Вы обнаруживаете, что MS Office на рабочем месте так и не обновили, а на вопросы об обновлении отвечают неопределенно. К тому же, некоторые из ваших документов предназначаются для других пользователей с устаревшими версиями Office - поэтому локальная самостоятельная установка конвертера на вашем ПК проблему не решает. А раз так - у вас есть два пути. Первый - заниматься *конверсией* файлов в формат .doc вручную, и второй - поручить это дело Microsoft Word.

**12.1.2. Решение**

Добавим в Microsoft Word кнопку, обработчик нажатия которой содержит следующий код ([листинг 12.1](https://www.intuit.ru/studies/courses/494/350/lecture/8369?page=1#example.12.1))

'Переменная для имени очередного файла

Dim var\_Doc As Variant

'Для хранения ссылки на открытый файл

Dim obj\_File As Document

'Для формирования нового имени файла

Dim str\_FileName As String

'Для подсчета количества обработанных файлов

Dim var\_FileCount

'Для хранения пути к папке с исходными файлами

Dim str\_In As String

'Папка с выходными файлами

Dim str\_Out As String

'Инициализируем переменные

str\_In = "C:\Документы"

str\_Out = "C:\Документы\Обработано"

'Изменяем текущую директорию на заданную

ChDir (str\_In)

'Ищем первый файл в директории

var\_Doc = Dir("\*.doc?")

Do While var\_Doc <> "" 'До тех пор, пока есть файлы

'Открываем найденный файл в невидимом окне

Set obj\_File = \_

Documents.Open( \_

FileName:=str\_In + "\" + var\_Doc, \_

Visible:=False)

'Формируем имя нового файла

str\_FileName = str\_Out + "\" + \_

Mid(var\_Doc, 1, Len(var\_Doc) - 4) + \_

"doc"

'Сохраняем файл в формате Word 97-2003

obj\_File.SaveAs FileName:=str\_FileName, \_

FileFormat:=wdFormatDocument97

'Закрываем файл

obj\_File.Close

'Увеличиваем на 1 счетчик файлов

var\_FileCount = var\_FileCount + 1

'Находим следующий подходящий файл в папке

var\_Doc = Dir()

Loop

MsgBox ("Обработано" + Str(var\_FileCount) + " файлов")

Листинг 12.1. Групповая обработка файлов

Как видите, алгоритм работы программы выглядит так:

* Найти файл
* Открыть его
* Сохранить в новом формате
* Закрыть

Эти шаги повторяются для каждого из файлов. Конструкции, использованные при решении этой задачи, мы рассматривали выше, поэтому у вас не должно возникнуть сложностей при разборе кода. Обратите внимание на то, что директории, пути к которым использованы в программе, должны быть созданы до ее запуска.

Вы можете доработать эту программу. Например, добавить сюда обработчик ошибок, дать пользователю возможность выбирать места для поиска исходных файлов и сохранения новых и т.д. Когда файл открыт, вы можете обрабатывать его любыми средствами Microsoft Word, то есть эта программа может послужить основой для множества программ, направленных на групповую работу с файлами.

**12.2. Копирование материалов нескольких документов в один**

[12-02-Копирование текстов.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/12/files/12_02.docm) - пример к п. 12.2.

**12.2.1. Условие**

Предположим, у вас есть несколько документов, материалы которых вы хотели бы скопировать в новый документ. Ядром этой задачи можно сделать описанную выше программу для *конверсии* файлов. Но здесь мы поступим по-другому. Предположим, что вы открыли все эти файлы и хотите быстро сбросить их содержимое в новый файл.

**12.2.2. Решение**

Для решения этой задачи напишем макрос, который должен храниться в глобальном шаблоне. При запуске макроса будет создаваться новый документ, в который будут скопированы материалы всех остальных открытых документов. Эту операцию можно реализовать множеством способов, мы избрали копирование материалов через буфер обмена Windows ([листинг 12.2](https://www.intuit.ru/studies/courses/494/350/lecture/8369?page=1#example.12.2).)

'Переменная для хранения ссылки на

'целевой документ

Dim obj\_TargetD As Document

'Для хранения ссылки на очередной

'обрабатываемый документ

Dim obj\_CurDoc As Document

Set obj\_TargetD = Documents.Add

For Each obj\_CurDoc In Documents

If obj\_CurDoc.Name <> \_

obj\_TargetD.Name Then

'Выделяем содержимое очередного документа

obj\_CurDoc.Content.Select

'Копируем в буфер обмена

Selection.Copy

'Активируем целевой документ

obj\_TargetD.Activate

'Вставляем в текущую позицию курсора

'материалы из буфера обмена

Selection.Paste

End If

Next

Листинг 12.2. Добавление материалов различных файлов в один файл

Здесь мы создаем новый документ, в который копируем содержимое остальных открытых документов. Цикл обхода всех объектов коллекции Documents, то есть открытых документов, позволяет получить доступ к каждому открытому документу. Однако, здесь не обойтись без проверки - мы должны обработать все открытые документы кроме того, в который копируем данные. Для этого прежде чем начать обработку, сравниваем имя целевого и текущего документа.

**12.3. Разбиение одного документа на несколько**

[12-03-Разбиение документа.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/12/files/12_03.docm) - пример к п. 12.3.

**12.3.1. Условие**

Задача, противоположная предыдущей. Один открытый документ нужно разбить на несколько, руководствуясь каким-либо признаком, после чего сохранить все документы, дав им соответствующие имена, в папку, заданную пользователем.

Разобъем активный документ на части, по 10 абзацев каждая, и сохраним каждую из частей в отдельном документе с именем такого вида:

Имя исходного документа+ "часть №"+номер части.docx

**12.3.2. Решение**

Напишем макрос ([листинг 12.3](https://www.intuit.ru/studies/courses/494/350/lecture/8369?page=1#example.12.3).), который имеет смысл хранить в глобальном шаблоне. Этот макрос будет работать с активным документом.

'Переменная для хранения ссылки на

'активный документ

Dim obj\_Doc As Document

'Для ссылки на новый документ

Dim obj\_NewDoc As Document

'Ссылка на абзац документа

Dim obj\_Par As Paragraph

'Строка для хранения пути сохранения

'файлов

Dim var\_Path As String

'Счетчик для подсчета обработанных

'абзацев

Dim var\_Counter

'Переменная для сборки имен

'новых файлов

Dim var\_FileName As String

'Переменная для хранения номера

'очередного файла, который используется

'при создании имени файла

Dim var\_Part

'Записываем 0 в счетчики количества

'абзацев и файлов и инициализируем

'другие переменные

var\_Counter = 0

var\_Part = 0

Set obj\_Doc = ActiveDocument

var\_Path = "C:\Документы"

'Цикл для каждого абзаца в документе

For Each obj\_Par In obj\_Doc.Paragraphs

'Если в предыдущем сеансе обработано

'10 абзацев - создается новый файл

'для записи в него очередных 10

If var\_Counter = 0 Then

Set obj\_NewDoc = Documents.Add

'Это - номер файла, который будет

'отражен в его имени

var\_Part = var\_Part + 1

End If

'Увеличиваем на 1 счетчик обработанных

'абзацев

var\_Counter = var\_Counter + 1

'Активируем новый документ

obj\_NewDoc.Activate

'Добавляем в него текст текущего абзаца

'из исходного документа

Selection.TypeText (obj\_Par.Range.Text)

'Если обработано 10 абзацев

If var\_Counter = 10 Then

'Сбрасываем счетчик обработанных абзацев

var\_Counter = 0

'Собираем имя нового файла

var\_FileName = var\_Path + "\" + \_

obj\_Doc.Name + \_

" часть № " + Str(var\_Part) + ".docx"

'Сохраняем новый файл

obj\_NewDoc.SaveAs FileName:=var\_FileName

'Закрываем его

obj\_NewDoc.Close

'Очищаем объектную переменную

Set obj\_NewDoc = Nothing

End If

Next

'Если в объектной переменной, представляющей

'новый файл, хранится ссылка

'это значит, что исходный файл имеет

'количество абзацев, не кратное 10,

'то есть он не будет сохранен в основном цикле

'сохраняем этот файл

If obj\_NewDoc Is Nothing = False Then

var\_FileName = var\_Path + "\" + \_

obj\_Doc.Name + \_

" часть № " + Str(var\_Part) + ".docx"

obj\_NewDoc.SaveAs FileName:=var\_FileName

obj\_NewDoc.Close

Set obj\_NewDoc = Nothing

End If

'Выводим сообщение об окончании работы

MsgBox ("Файл " + obj\_Doc.Name + \_

" обработан, получено " + \_

Str(var\_Part) + " файлов, которые" + \_

" хранятся в: " + var\_Path)

Листинг 12.3. Разбиваем документ на части

Здесь мы работаем с исходным документом, поабзацно копируя его содержимое в новый документ. В программе использована пара переменных-счетчиков. Одна служит для подсчета количества обработанных абзацев - она используется как индикатор для сохранения или создания очередного файла, а вторая выступает счетчиком номеров созданных файлов и участвует в создании имени файла.

**12.4. Точная замена**

[12-04-Точная замена.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/12/files/12_04.docm) - пример к п. 12.4.

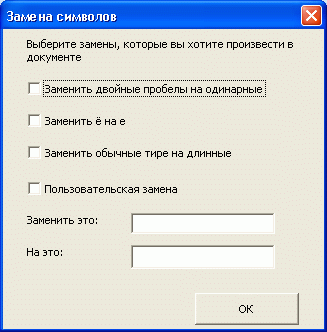
Довольно часто, при работе с текстами возникает необходимость их проверки и обработки некоторых символов и их последовательностей. Например, это касается удаления двойных пробелов, замены обычного тире на длинное, замены буквы е на ё, и т.д. Все эти замены можно сделать с помощью средства Найти и заменить. Однако, несложно заметить, что даже несколько таких замен отнимают много времени - нужно ввести данные в окно, дождаться выполнения *операции*, ввести другие данные. Логично было бы автоматизировать этот процесс.

**12.4.1. Условие**

Написать программу на VBA, которая позволяла бы автоматизировать процесс замены одних комбинаций символов другими. В частности, предусмотреть выбор среди следующих объектов замены: заменить букву **е** на **ё**, удалить двойные пробелы, заменить обычное тире на длинное. Предусмотреть возможность задавать один из объектов пользователем.

**12.4.2. Решение**

Создадим пользовательскую форму ([рис. 12.1](https://www.intuit.ru/studies/courses/494/350/lecture/8369?page=2#image.12.1).).



**Рис. 12.1.**Форма для настройки замены символов

Основные элементы управления, которые мы используем, это следующие флажки ([табл. 12.1](https://www.intuit.ru/studies/courses/494/350/lecture/8369?page=2#table.12.1).).

|  |  |
| --- | --- |
| Таблица 12.1. Основные элементы управления | |
| **Имя элемента управления** | **Подпись** |
| chk\_Space | Заменить двойные пробелы на одинарные |
| chk\_Yo | Заменить букву е на ё |
| chk\_LongDash | Заменить обычные тире на длинные |
| chk\_User | Пользовательская замена |
| cmd\_OK | ОК |

Если выбран флажок chk\_User - активируются два текстовых поля ( txt\_One и txt\_Two ). Их свойства Enabled первоначально установлены в False, при выборе флажка оно устанавливается в True, при снятии - обратно в False. В поле txt\_One пользователь может записать последовательность символов, которую нужно заменить на последовательность, которую он может ввести в txt\_Two.

Ниже приведен код *пользовательских процедур* и обработчиков *событий элементов управления*, которые реализуют вышеописанную функциональность. [Листинг 12.4](https://www.intuit.ru/studies/courses/494/350/lecture/8369?page=2#example.12.4). содержит код обработки события Change для chk\_User.Листинг 12.5. представляет собой код, который выполняется при нажатии на кнопку ОК, которая запускает процесс замен. [Листинг 12.6](https://www.intuit.ru/studies/courses/494/350/lecture/8369?page=2#example.12.6). содержит код процедуры, которая вызывает стандартное средство для замены.

Private Sub chk\_User\_Change()

'Выполняется при изменении состояния

'флажка chk\_User

If chk\_User = True Then

'Если флажок установлен 'активировать поля

txt\_One.Enabled = True

txt\_Two.Enabled = True

Else

'Если флажок снят 'заблокировать поля

txt\_One.Enabled = False

txt\_Two.Enabled = False

End If

End Sub

Листинг 12.4. Обработка события Change для chk\_User

Private Sub cmd\_Ok\_Click()

'Обработчик нажатия кнопки

'Проверяем, установлен ли каждый

'из флажков и в зависимости от этого

'выполняем вызов пользовательской процедуры

'которая содержит код вызова средства

'Найти и заменить

If chk\_Space = True Then

Call ReplaceThis(" ", " ")

End If

If chk\_Yo = True Then

Call ReplaceThis("е", "е")

End If

If chk\_LongDash = True Then

Call ReplaceThis("-", "-")

End If

If chk\_User = True Then

Call ReplaceThis(txt\_One, txt\_Two)

End If

End Sub

Листинг 12.5. Обработка события нажатия на кнопку cmd\_OK

Public Sub ReplaceThis(str\_1, str\_2 As String)

'Пользовательская процедура

'Принимает два параметра

'str\_1 - это символ, который надо заменить

'str\_2 - символ, на который надо заменить

'Поиск и замена во всем документе

ActiveDocument.Content. \_

Find.Execute \_

FindText:=str\_1, ReplaceWith:=str\_2, \_

Replace:=wdReplaceAll

End Sub

Листинг 12.6. Процедура вызова стандартного средства поиска и замены символов

Программа, которую мы написали в этом примере, может использоваться на практике в своем исходном виде. Однако, скорее всего вам захочется дополнить ее собственными символами для замены.

**12.5. Проверка правильности создания документа**

[12-05-Проверка правильности.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/12/files/12_05.docm) - пример к п. 12.5.

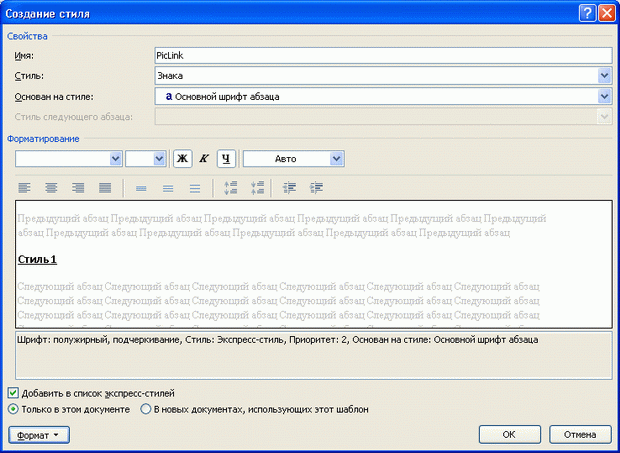
Этот пример будет посвящен использованию стилей при программной обработке документов. Стиль - это набор правил форматирования документа, имеющий имя.

**12.5.1. Условие**

Предположим, что в нашем документе используется стиль с именем PicLink для форматирования ссылки на изображение в тексте, и стиль PicName - для самого имени изображения. Изображение должно следовать в документе сразу после его упоминания в тексте. Нужно написать программу, которая проверяет номера изображений, упомянутых в тексте и номера иллюстраций и составляет после проверки отчет о соответствии ссылок на иллюстрации их реальным номерам. Для нумерования иллюстраций используется конструкция такого вида: "Рис. X".

**12.5.2. Решение**

Для начала создадим два стиля. Один назовем PicLink, второй - PicName. Для создания стиля отформатируйте какой-нибудь участок документа так, как нужно, после чего выделите его, щелкните по выделению правой кнопкой мыши и выберите в появившемся меню пункт Стили o Сохранить выделенный фрагмент как новый экспресс-стиль. Появится окно для настройки свойств стиля, разверните его кнопкой Изменить ([рис. 12.2](https://www.intuit.ru/studies/courses/494/350/lecture/8369?page=2#image.12.2).). Задайте имя стиля, место сохранения, в поле Стиль выберите Знака, и нажмите кнопку OK.

[](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/12/files/12_02.gif)

[увеличить изображение](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/12/files/12_02.gif)  
**Рис. 12.2.**Создание стиля

В [листинге 12.7](https://www.intuit.ru/studies/courses/494/350/lecture/8369?page=2#example.12.7). вы можете найти код, с помощью которого реализуется функциональность, о которой идет речь в условии задачи. Этот макрос просматривает документ и проверяет, следует ли за упоминанием иллюстрации в тексте заголовок иллюстрации. В ходе проверки программа составляет отчет, который она выгружает в отдельный файл.

'Для хранения ссылки на символ

Dim obj\_Char As Range

'Для хранения текста ссылки на

'изображение

Dim str\_PicLink As String

'Для хранения текста названия

'изображения

Dim str\_PicName As String

'Для хранения текста отчета

Dim str\_ResultText

'Счетчик иллюстраций

Dim num\_Counter

'Ссылка на документ с отчетом

Dim obj\_Doc As Document

'Для каждого символа в документе

For Each obj\_Char In ActiveDocument.Characters

'Если стиль символа - PicLink

'и символ является цифрой, точкой или

'косой чертой - добавляем символ

'в переменную

If obj\_Char.Style = "PicLink" Then

If Asc(obj\_Char.Text) >= 46 And \_

Asc(obj\_Char.Text) <= 57 Then

str\_PicLink = str\_PicLink + \_

obj\_Char.Text

End If

End If

'Аналогично - для стиля PicName

If obj\_Char.Style = "PicName" Then

If Asc(obj\_Char.Text) >= 46 And \_

Asc(obj\_Char.Text) <= 57 Then

str\_PicName = str\_PicName + \_

obj\_Char.Text

End If

End If

'Если стиль - не PicLink и не PicName

'и переменные, хранящие симолы, взятые

'из ссылки на рисунок и названия, не

'пусты - проверяем, одинаковы ли эти

'символы и формируем строку для

'последующего вывода в файл отчета

If obj\_Char.Style <> "PicLink" And \_

obj\_Char.Style <> "PicName" And \_

Len(str\_PicLink) > 0 And \_

Len(str\_PicName) > 0 Then

'Увеличим на 1 счетчик изображений

num\_Counter = num\_Counter + 1

If str\_PicLink = str\_PicName Then

str\_ResultText = str\_ResultText + \_

"Иллюстрация №" + Str(num\_Counter) + \_

" " + str\_PicLink + " верно" + \_

" пронумерована" + Chr(13)

Else

str\_ResultText = str\_ResultText + \_

"Внимание! Иллюстрация №" + \_

Str(num\_Counter) + \_

" " + str\_PicLink + " неверно" + \_

" пронумерована" + Chr(13)

End If

'Сбросим переменные для хранения

'новой информации

str\_PicLink = ""

str\_PicName = ""

End If

Next

'сформируем файл отчета

Set obj\_Doc = Documents.Add

obj\_Doc.Activate

Selection.TypeText str\_ResultText

Листинг 12.7. Макрос для проверки правильности документа

**12.6. Заключение**

Мы рассмотрели множество вопросов программирования для Microsoft *Word*. Теперь вы сможете сделать с помощью этого редактора много полезного.

Пришло время поговорить о MS *Excel*. Помимо программирования для *Excel*, в следующей теме мы рассмотрим такие важные вопросы, как взаимодействие приложений и работа с базами данных.

Лекция 13. Объектная модель MS Excel, объект Application

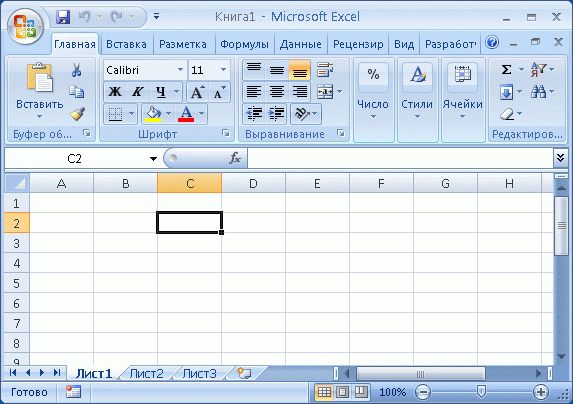
**13.1. Особенности программирования для MS Excel**

Microsoft Office *Excel* - это популярные электронные таблицы. Обычно, программируя для этой программы, преследуют такие цели:

* Автоматизация вычислений.
* Автоматизация ввода и обработки информации.
* Работа с базами данных - вывод, ввод, анализ, визуализация информации.
* Анализ финансовой и другой информации.
* Создание систем для организации автоматизированного ввода данных
* *Математическое моделирование*.

В общем виде *программирование* в *Excel* похоже на работу с Microsoft *Word*. Однако одно из главных отличий заключается в том, что в *Excel* рабочая область листа разбита на ячейки, каждая из которых имеет собственное имя. Имена ячеек могут быть двух видов.

* Первый вид (стиль A1) - это имя, состоящее из буквенного имени столбца и номера строки. Например, A1 - ячейка, находящаяся на пересечении столбца А (первого) и первой строки.
* Другой вид - это индексы ячеек (стиль именования R1C1 ). Для адресации ячейки в таком стиле указывают номер строки (R - Row - строка) и номер столбца (C - Column - столбец), на пересечении которых расположена ячейка. Строки изначально пронумерованы, а номера столбцов начинаются с 1 - первому столбцу соответствует столбец A, второму - В и т.д. Например, (2, 3) - это адрес ячейки, находящийся на пересечении второй строки и третьего столбца, то есть, если переложить это на стиль A1 - получим ячейку С2 ([рис. 13.1](https://www.intuit.ru/studies/courses/494/350/lecture/8371?page=1#image.13.1).)



**Рис. 13.1.**Ячейка C2 на листе MS Excel

Для выполнения большинства операций в MS *Excel* применяются следующие объекты.

* Excel.Application (Приложение) - объект, представляющий приложение Microsoft Excel, аналогичен Word.Application.
* *Workbook* (*Рабочая книга*) - представляет *рабочую книгу* - аналог документа Microsoft Word. Однако, в Word мы работаем с данными, расположенными в документе, а в Excel на пути к данным есть еще один объект - рабочий лист.
* *Worksheet* (Рабочий лист) - книга в MS Excel разбита на рабочие листы. Именно на листе расположены ячейки, которые могут хранить информацию и формулы.
* Range (Диапазон) - может быть представлен в виде одной ячейки или группы ячеек. Этот объект заменяет множество объектов для работы с элементами документов ( character, word и т.д.), которые применяются в Microsoft Word. В результате работа с листом становится очень четкой и удобной - чтобы работать с какой-либо ячейкой, надо знать лишь ее имя (в формате A1) или адрес ( R1C1 ).
* QueryTable (Таблица запросов) - этот объект используют для импорта в Microsoft Excel информации из баз данных. *Подключение к базе данных*, запрос информации и т.д. производятся средствами объекта, а итоги запроса выгружаются на лист MS Excel в виде обычной таблицы.
* PivotTable (Сводная таблица) - это особый вид электронной таблицы Excel - она позволяет в интерактивном режиме обобщать и анализировать большие объемы информации, в частности, взятой из базы данных.
* *Chart* (Диаграмма) - представляет собой диаграмму. Обычно их используют для визуализации данных.

Давайте начнем рассмотрение *объектной модели MS Excel* с объекта Application.

**13.2. Объект Application**

[13-01-Excel Application.xlsm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/13/files/13_01.xlsm) - пример к п. 13.2.

Принципы работы с объектом Application в MS *Excel* очень похожи на таковые в MS *Word*.

Рассмотрим пример. Откроем MS *Excel*, добавим на *рабочий лист* кнопку, добавим в обработчик щелчка по кнопке такой код ([листинг 13.1](https://www.intuit.ru/studies/courses/494/350/lecture/8371?page=1#example.13.1).).

MsgBox Excel.Application.Name

Листинг 13.1. Вывести имя приложения

После выполнения программы в окне сообщения отобразится имя приложения - в данном случае - Microsoft *Excel*. Очевидно, что *свойство*Name объекта Application возвращает имя приложения.

Теперь рассмотрим наиболее важные *методы*и *свойства* Application. Некоторые из них похожи на таковые в MS *Word*. Например, метод Quit точно так же, как в *Word*, закрывает *приложение*, свойство Visible отвечает за видимость окна программы и т.д.

**13.3. Методы Application**

**13.3.1. Calculate - принудительный пересчет**

Этот метод, вызываемый для объекта Application, позволяет пересчитать все открытия книги. Его же можно вызывать для отдельных книг (объект *Workbook* ) листов ( *Worksheet* ), ячеек и их диапазонов ( Range ). Например, код из [листинга 13.2](https://www.intuit.ru/studies/courses/494/350/lecture/8371?page=1#example.13.2). позволяет пересчитать все открытые книги.

Application.Calculate

Листинг 13.2. Пересчитать все открытые книги

**13.3.2. GoTo - переход в ячейку**

[13-02-Excel GoTo.xlsm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/13/files/13_02.xlsm) - пример к п. 13.3.2.

Позволяет выделить любой *диапазон ячеек* в любой книге, причем, если книга не активна - она будет активирована. Так же метод может запускать макросы Microsoft Excel.

Полный вызов метода выглядит так:

Goto(Reference, Scroll)

В качестве параметра Reference используется ссылка на ячейку или *диапазон ячеек*, которые должны быть выделены после вызова метода. Так же здесь может быть использовано имя макроса - тогда он будет запущен.

Параметр Scroll отвечает за "перемотку" листа Excel к выделенным ячейкам - так, чтобы левый верхний угол выделения совпадал бы с левым верхним углом отображаемой области листа. Если Scroll установлен в True - лист перематывается, если в False - нет.

Например, такой вызов ([листинг 13.3](https://www.intuit.ru/studies/courses/494/350/lecture/8371?page=1#example.13.3).) позволяет выделить ячейку H500 на активном листе.

Application.Goto \_

Reference:=ActiveSheet.Range("H500"), \_

Scroll:=True

Листинг 13.3. Выделить ячейку H500

Как видите, обращение к активному листу очень напоминает обращение к активному документу в MS Word. Обратите внимание на то, что мы используем полное обращение к методу - Application.GoTo - как вы знаете, обычно свойства и методы объекта Application можно использовать в коде, не указывая этого объекта. Однако, если в этом случае не указать Application, то вместо метода GoTo программа попытается выполнить *оператор безусловного перехода* GoTo.

**13.3.3. SendKeys - имитация нажатий на клавиши клавиатуры**

[13-03-Excel SendKeys.xlsm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/13/files/13_03.xlsm) - пример к п. 13.3.3.

Очень интересный метод - позволяет передавать в активное окно приложения нажатия клавиш. Полный вызов метода выглядит так:

SendKeys(Keys, Wait)

Параметр Keys позволяет задавать клавиши, нажатия которых будут переданы приложению. Метод поддерживает эмуляцию как алфавитно-цифровых, так и *управляющих клавиш*, для которых применяется специальная кодировка. Алфавитно-цифровые клавиши указываются при вызове в своем обычном виде Например, для передачи символа "Ф" нужно указать его при вызове метода и т.д.

Чтобы передать приложению нажатия клавиши **Backspace** - используйте код {BS}. Для передачи нажатия кнопки **Enter** используйте значок ~ (*тильда*), для клавиши **Del** - код {DEL}.

Для передачи приложению нажатий клавиш Shift, Ctrl или Alt, можно воспользоваться следующими кодами этих клавиш:

* **SHIFT** - + (знак плюс)
* **CTRL -** ^ (крышка)
* **ALT**- % (знак процента)

Параметр Wait может принимать значения True или False. Если он будет установлен в True - макрос подождет, пока приложение обработает введенные с "клавитуры" данные, после чего продолжит выполнение. Если в False - макрос ждать не будет.

Рассмотрим пример. С помощью метода SendKeys введем в ячейку A1 текст "Сейчас закрою программу!", после чего попытаемся закрыть программу, "нажав" сначала Alt, потом Ф (для открытия главного меню Office), и потом - Ы - для выхода из программы. Подразумевается, что при запуске макроса ([листинг 13.4](https://www.intuit.ru/studies/courses/494/350/lecture/8371?page=1#example.13.4).) указанная ячейка активна - в ней находится рамка выделения.

Application.SendKeys ("Сейчас закрою программу!")

Application.SendKeys ("%ФЫ")

Листинг 13.4. Имитация нажатий клавиш клавиатуры

После выполнения этого кода вы увидите окно со стандартным вопросом о сохранении документа.

**13.3.4. Wait - пауза при выполнении макроса**

[13-04-Excel Wait.xlsm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/13/files/13_04.xlsm) - пример к п. 13.3.4.

Позволяет сделать паузу в выполнении макроса. Пауза может быть нужна для того, чтобы пользователь успел что-то рассмотреть или для того, чтобы на время освободить вычислительные ресурсы системы.

При вызове этого метода указывается время, на которое должно быть приостановлено выполнение макроса. После того, как истекает время, заданное при вызове метода, он возвращает True и выполнение программы продолжается. Например, в [листинге 13.5](https://www.intuit.ru/studies/courses/494/350/lecture/8371?page=1#example.13.5). так можно остановить выполнение прораммы на 7 секуд, после чего - вывести сообщение.

If Application.Wait(Now + TimeValue("0:00:7")) Then

MsgBox "Семь секунд прошло!"

End If

Листинг 13.5. Приостановка выполнения программы

Здесь мы получаем значение типа Date с помощью функции TimeValue. После того, как пройдут 7 секунд, выполнение макроса возобновится, метод возвратит True, будет выполнено условие и выведено сообщение.

**13.4. Свойства Application**

**13.4.1. ActiveCell, ActiveChart, ActivePrinter, ActiveSheet, ActiveWindow, ActiveWorkbook - активные объекты**

[13-05-Excel Active.xlsm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/13/files/13_05.xlsm) - пример к п. 13.4.1.

Cвойства, имена которых начинаются с Active, позволяют обращаться к различным активным объектам.

ActiveCell возвращает объект типа Range, который представляет собой активную (выделенную) ячейку рабочего листа, отображаемого в данный момент на экране. Если при вызове этого свойства на экране нет открытого листа - произойдет ошибка.

Например, такой код ([листинг 13.6](https://www.intuit.ru/studies/courses/494/350/lecture/8371?page=2#example.13.6).) выводит данные из *активной ячейки* в окне сообщения, после чего предлагает пользователю ввести в эту ячейку новые данные с помощью окна ввода.

MsgBox ("В ячейке с именем " + Application.ActiveCell.Address + " хранится значение " + & Application.ActiveCell.Value)

ActiveCell.Value = InputBox("Введите новое значение для ячейки " + ActiveCell.Address)

Листинг 13.6. Работа с активной ячейкой

Очевидно, что свойство Value объекта ActiveCell содержит данные, которые записаны в ячейку, а свойство Address - адрес ячейки.

Остальные свойства этой группы предназначены для обращения к следующим объектам:

* ActiveChart - к активной диаграмме.
* ActivePrinter - к активному принтеру.
* ActiveSheet - к активному листу. Это свойство очень часто используется на практике. Например, [листинг 7.7](https://www.intuit.ru/studies/courses/494/350/lecture/8371?page=2#example.7.7). позволяет вывести имя активного листа.
* MsgBox Application.ActiveSheet.Name
* ActiveWindow - к активному окну.

ActiveWorkbook - к активной рабочей книге.

Листинг 7.7. Выводим имя активного листа

**13.4.2. Cells, Columns, Rows, Sheets, Workbooks, Worksheets, Names - наборы объектов и коллекции**

Эти свойства возвращают соответствующие наборы объектов и коллекции. Подробности о них мы рассмотрим ниже, здесь лишь определим их основное предназначение.

* Cells, Columns, Rows - возвращают наборы объектов Range, содержащие, соотвественно, ячейки, столбцы, строки. При вызове этих свойств можно указывать, какие именно объекты нужно возвратить, а можно, вызвав без параметров, получить все объекты нужного вида.
* Sheets, *Worksheets* - возвращают коллекции, которые содержат листы активной книги. В коллекции Sheets будут содержаться листы, которые содержат диаграммы и обычные листы, а в коллекции *Worksheets* - лишь обычные листы.
* *Workbooks* - возвращает коллекцию открытых книг.
* Names - возвращает коллекцию именованных диапазонов - с ними можно работать так же, как с закладками в MS Word.

**13.4.3. Range - ячейка или группа ячеек**

Возвращает объект Range, который ссылается на ячейку или группу ячеек. Это - один из важнейших объектов для работы с ячейками - ниже мы остановимся на нем подробнее.

**13.4.4. ScreenUpdating - обновление экрана**

[13-06-Application ScreenUpdating.xlsm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/13/files/13_06.xlsm) - пример к п. 13.4.4.

Позволяет включать (присвоением свойству True ) и отключать (присвоением False ) обновление экрана. Имеет смысл отключить обновление экрана перед теми частями программы, которые интенсивно пользуются данными на листе. Благодаря тому, что системные ресурсы не будут тратиться на обновление экрана, программа будет работать быстрее. Этот метод весьма актуален, так как MS Excel часто используют для проведения ресурсоемких расчетов.

Практика показывает, что если программа интенсивно использует вывод на экран в процессе работы, если она изменяет данные, которые участвуют в расчете формул, расположенных на листе, то отключение вывода может ускорить работу в 3-10 раз.

Например, ниже ([листинг 13.8](https://www.intuit.ru/studies/courses/494/350/lecture/8371?page=2#example.13.8).) приведен код, который два раза повторяет процедуру 100-кратного вывода на экран 400 целых случайных чисел и выводит время, требующееся для выполнения этих действий с обновлением экрана и без него.

'Массив для значений времени

Dim WorkTime(2)

'Время начала теста

Dim StartTime

'Время окончания теста

Dim StopTime

'Включаем обновление

Application.ScreenUpdating = True

For i = 1 To 2

'Во втором проходе цикла

'выключим обновление

If i = 2 Then \_

Application.ScreenUpdating = False

End If

'Запишем текущее время

StartTime = Time

'Перейдем на лист для теста

Worksheets("Тест скорости").Activate

'Выведем 100 раз целые случайные

'числа в область 20х20

For y = 1 To 100

For p = 1 To 20

For j = 1 To 20

ActiveSheet.Cells(p, j) = \_

Int(Rnd \* 100)

Next j

Next p

Next y

'Запишем время окончания

StopTime = Time

'Для корректного представления

'в виде секунд

WorkTime(i) = \_

(StopTime - StartTime) \* 24 \* 60 \* 60

Next i

Application.ScreenUpdating = True

MsgBox "Время выполнения программы." & Chr(13) + \_

"При включенном обновлении: " & \_

Round(WorkTime(1),2) & " сек." & Chr(13) & \_

"При выключенном обновлении: " & \_

Round(WorkTime(2),2) & " сек."

Листинг 13.8. Оценка скорости работы с обновлением экрана и без него

**13.4.5. Selection - ссылка на выделенный объект**

Это очень важное свойство возвращает ссылку на выделенный объект. Чаще всего это - ячейка или группа ячеек. Например, это свойство удобно использовать при работе с выделенным *диапазоном ячеек* (или отдельной выделенной ячейкой). Ниже мы коснемся его подробнее.

**13.4.6. WorksheetFunction - формулы Excel в коде VBA**

Возвращает объект WorksheetFunction, методы которого представляют собой формулы Excel, которые можно использовать в коде VBA. Использование этого свойства позволяет облегчить выполнение сложных расчетов.

**13.5. События Application**

[13-07-Excel Application Events.xlsm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/13/files/13_07.xlsm) - пример к п. 13.5.

*Объект* Excel.Application поддерживает множество *событий*. Работа с ними аналогична работе с событиями Word.Application, которыми мы занимались в соответствующем разделе предыдущей главы.

Рассмотрим основные шаги, которые необходимо произвести, чтобы работать с событиями приложения, перечислим события и приведем пример.

Создайте новый *модуль* класса. Добавьте в него объявление объекта типа Excel.Application с событиями ([листинг 13.9](https://www.intuit.ru/studies/courses/494/350/lecture/8371?page=2#example.13.9).).

Public WithEvents obj\_ExApp As Excel.Application

Листинг 13.9. Объявляем новый объект типа Excel.Application с событиями

После этого в списке объектов редактора кода модуля появится *объект* obj\_ExApp, а в списке событий - соответствующие ему события. Выберите нужное вам событие - автоматически будет создан обработчик для него. В частности, Excel.Application поддерживает следующие события:

* NewWorkbook - происходит при создании новой книги
* SheetActivate - при активации любого листа
* SheetBeforeDoubleClick - происходит при двойном щелчке по листу, то есть позволяет перехватить щелчок и выполнить собственную процедуру до того, как будет выполнено стандартное действие.
* SheetBeforeRightClick - позволяет перехватить нажатие правой кнопки мыши по листу.
* SheetCalculate - после пересчета листа или после изменения данных, которые отображаются на диаграмме.
* SheetChange - при изменении содержимого ячеек на любом листе.
* SheetFollowHyperlink - происходит при переходе по гиперссылке, которая может быть включена в лист Microsoft Excel.
* SheetSelectionChange - при изменении выделения на листе
* WindowActivate - при активации окна книги.
* WindowDeactivate - при деактивации окна книги.
* WindowResize - при изменении размера окна книги.
* WorkbookActivate - при активации книги.
* WorkbookBeforeClose - перед закрытием книги.
* WorkbookBeforePrint - перед печатью книги.
* WorkbookBeforeSave - перед сохранением книги.
* WorkbookDeactivate - при деактивации книги.
* WorkbookNewSheet - при добавлении нового листа в любую из открытых книг.
* WorkbookOpen - при открытии книги.

После того, как создан обработчик, написан его код, работа еще не окончена. Следующий шаг - это *связывание* объекта obj\_ExApp с реально работающим приложением. Ниже приведен полный код модуля с одним обработчиком события, а также - процедура, служащая для связывания объекта obj\_ExApp с работающим приложением. Эта процедура может существовать в виде отдельного макроса или в виде кода обработчика нажатия на кнопку. Ее выполнение можно назначить событию открывающейся книги, которая содержит данный *модуль* класса и т.д.

Итак, вот ([листинг 13.10](https://www.intuit.ru/studies/courses/494/350/lecture/8371?page=2#example.13.10).) код процедуры, который связывает *объект* созданного нами класса AppEvents с приложением:

Dim obj\_ExcelAppEv As New AppEvents

Sub EventsInit()

Set obj\_ExcelAppEv.obj\_ExApp = Excel.Application

End Sub

Листинг 13.10. Связываем объект с приложением

А вот ([листинг 13.11](https://www.intuit.ru/studies/courses/494/350/lecture/8371?page=2#example.13.11).) полный код модуля класса AppEvents с объявлением объектной переменной и обработчиком события.

Public WithEvents obj\_ExApp As Excel.Application

Private Sub obj\_ExApp\_NewWorkbook(ByVal Wb As Workbook)

'Выполняется при создании новой книги

MsgBox "Вы создали новую книгу"

End Sub

Листинг 13.11. Код модуля класса с обработчиком события

**13.6. Выводы**

В этой лекции мы обсудили методы, свойства и события объекта Application приложения Microsoft *Excel*. Можно заметить, что некоторые из этих методов прямо указывают на то, что *Excel* - это *приложение*, рассчитанное на проведение достаточно серьезных расчетов. В следующей лекции мы рассмотрим особенности работы с документами MS *Excel*.

Лекция 14. Работа с книгами и листами

**14.1. Коллекция Workbooks - работа с книгами**

Основные *операции*, которые приходится выполнять при работе с *книгами* - это их автоматическое создание, выбор одной из открытых книг для обработки, открытие *книги*. Все открытые *книги* принадлежат *коллекции* *Workbooks*.

Практически все часто используемые методы и свойства *коллекции* *Workbooks* очень похожи на соответствующие методы и свойства коллекции Documents из объектной модели MS *Word*.

**14.2. Методы Коллекции Workbooks**

**14.2.1. Add, Close, Open - стандартная работа с книгами**

[14-01-Excel Workbooks Add.xlsm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/14/files/14_01.xlsm) - пример к п. 14.2.1.

* Add - создает новую *книгу*. Новая *книга* становится активной.
* Close - закрывает все открытые *книги*.
* Open - открывает указанную *книгу*. Основной параметр метода - имя открываемой *книги*.

Например, такой код ([листинг 14.1](https://www.intuit.ru/studies/courses/494/350/lecture/8372?page=1#example.14.1).) позволяет создать новую *книгу*, присвоив ссылку на нее *объектной переменной*.

Dim obj\_Workbook As Workbook

Set obj\_Workbook = Workbooks.Add

Листинг 14.1. Создаем новую книгу

**14.2.2. OpenText, OpenXML - открытие различных файлов**

Эти методы предназначены для открытия текстовых файлов ( OpenText ) и XML-файлов ( OpenXML ) в виде книг MS Excel.

**14.3. Свойства коллекции Workbooks**

[14-02-Excel Workbooks Properties.xlsm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/14/files/14_02.xlsm) - пример к п. 14.3.

**14.3.1. Count**

Возвращает количество объектов в коллекции. В данном случае - количество открытых книг.

**14.3.2. Item**

Позволяет обращаться к выбранному элементу (отдельной *книге* ) в коллекции. В качестве идентификатора *книги* может использоваться ее индекс или имя.

В [листинге 14.2](https://www.intuit.ru/studies/courses/494/350/lecture/8372?page=1#example.14.2). мы сначала выводим информацию о количестве книг, а потом - отображаем имя каждой открытой *книги*.

MsgBox "Открыто" + Str(Workbooks.Count) + \_

" книг"

For i = 1 To Workbooks.Count

MsgBox Workbooks.Item(i).Name + \_

" имя открытой книги №" + Str(i)

Next i

Листинг 14.2. Вывод информации об открытых книгах

**14.4. Объект Workbook**

Workbook - *книга* - это аналог документа ( Document ) в Microsoft *Word*. Многие свойства и методы *Workbook* действуют аналогично свойствам объекта Document. Однако *книга* в объектной модели состоит из *листов*. Значит, для обращения к данным, хранящимся в *книге*, нужно сначала обратиться к *листу*.

**14.5. Методы объекта Workbook**

Методы *объекта* *Workbook* очень похожи на методы объекта Document в MS *Word*. Поэтому здесь мы лишь перечислим их.

**14.5.1. Activate, Close, Save, SaveAs, PrintOut**

* Activate - активирует *книгу*
* Close - закрывает *книгу*
* Save - сохраняет *книгу*
* SaveAs - сохраняет *книгу* с новым именем, в новом формате и т.д.
* *PrintOut* - выводит *книгу* на печать

**14.6. Свойства объекта Workbook**

**14.6.1. Свойства ActiveChart, ActiveSheet, Charts, Sheets, Worksheets, Styles, Names**

Эти свойства возвращают различные объекты и коллекции объектов. В частности, ActiveChart возвращает активную диаграмму, ActiveSheet - активный лист *книги*, а *Charts* и Sheets возвращают, соответсвенно, коллекции *листов*, содержащих диаграммы, и всех *листов* *книги*. В объект *Workbook* входят и другие подобные коллекции - они очень похожи на коллекции, входящие в Application.

Чаще всего, работая с *объектом* *Workbook*, обращаются к коллекциям Sheets и *Worksheets* - ведь именно они дают доступ к *рабочим листам*.

**14.6.2. Свойства CodeName, FullName, Name**

[14-03-Excel Workbook Name.xlsm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/14/files/14_03.xlsm) - пример к п. 14.6.2.

Эти свойства возвращают различные имена *книги*. Так, CodeName - это имя, под которым *книга* видна в окне Project Explorer (обычно ЭтаКнига ). FullName - имя файла *книги* вместе с путем. Name - имя файла *книги*.

В [листинге 14.3](https://www.intuit.ru/studies/courses/494/350/lecture/8372?page=1#example.14.3). вы можете найти пример вывода различных имен *книги*.

MsgBox "Кодовое имя текущей книги: " + \_

ActiveWorkbook.CodeName + Chr(13) + \_

"Имя текущей книги: " + \_

ActiveWorkbook.Name + Chr(13) + \_

"Полное имя книги: " + \_

ActiveWorkbook.FullName

Листинг 14.3. Вывод различных имен книги

**14.7. События объекта Workbook**

[14-04-Excel Workbook Events.xlsm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/14/files/14_04.xlsm) - пример к п. 14.7.

В VBA-проекте, который прикреплен к *книге*, есть *объект* ЭтаКнига. Если сделать по нему *двойной щелчок* - будет открыто окно редактора кода, в котором, по обычной схеме, можно выбрать *объект* - в данном случае это *Workbook*, а также события, для которых автоматически будет создан обработчик.

Книги в Microsoft *Excel* могут реагировать на *множество событий*, среди них следующие, которые происходят при таких обстоятельствах:

* Activate - при активации *книги*
* BeforeClose - перед закрытием *книги*, если *книга* изменялась и должен появиться запрос на ее сохранение - происходит перед сохранением.
* BeforePrint - перед печатью.
* BeforeSave - перед сохранением.
* Deactivate - при деактивации.
* NewSheet - при добавлении нового *листа* в *книгу*.
* Open - при открытии *книги*.
* SheetActivate - при активации любого *листа*
* SheetBeforeDoubleClick - после двойного щелчка в *книге*, до стандартной процедуры обработки.
* SheetBeforeRightClick - перед щелчком правой кнопки мыши.
* SheetCalculate - при пересчете *листа* или изменении данных, которые отображаются на диаграмме.
* SheetChange - при изменении содержимого ячейки.
* SheetDeactivate - при деактивации *листа*.
* SheetFollowHyperlink - при щелчке по гиперссылке.
* SheetSelectionChange - при изменении выделения.

Например, [листинг 14.4](https://www.intuit.ru/studies/courses/494/350/lecture/8372?page=1#example.14.4). позволяет вывести в окне сообщения информацию об имени измененной ячейки, имени *листа*, на котором она расположена, и новом содержимом ячейки.

Private Sub Workbook\_SheetChange(ByVal Sh As Object, \_

ByVal Target As Range)

MsgBox "Вы отредактировали даные на листе: " & \_

Sh.Name & ", в ячейке " & Target.Address & \_

". Теперь ячейка содержит следующие данные: " & \_

Target.Value

End Sub

Листинг 14.4. Вывод информации об измененной ячейке

Здесь мы воспользовались параметрами, которые передаются в обработчик события. *Параметр* Sh содержит ссылку на *лист*, на котором произошли изменения, а Target представляет собой ссылку на измененную ячейку или *диапазон*.

**14.8. Коллекция Worksheets**

*Коллекция Worksheets* содержит *листы* открытой *книги*. Ее свойства и методы служат для работы с ними.

**14.9. Методы коллекции Worksheets**

**14.9.1. Add - добавляем листы в книгу**

[14-05-Excel Worksheets Add.xlsm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/14/files/14_05.xlsm) - пример к п. 14.9.1.

Позволяет добавить в *книгу* новый лист. Полный вызов метода выглядит так.

Add(Before, After, Count, Type)

Before (До) и After (После) позволяют задавать позицию *листа* относительно других *листов* *книги* (можно использовать лишь один из них), Count задает количество добавляемых *листов*, а Type - тип *листа*. Чаще всего этот метод используют без параметров.

Например, [листинг 14.5](https://www.intuit.ru/studies/courses/494/350/lecture/8372?page=1#example.14.5). добавляет новый лист в *книгу*, в которой хранится макрос, содержащий этот код:

ThisWorkbook.Worksheets.Add

Листинг 14.5. Добавляем новый лист

А так можно добавить в *книгу* столько *листов*, сколько указал пользователь в ответ на вопрос программы ([листинг 14.6](https://www.intuit.ru/studies/courses/494/350/lecture/8372?page=1#example.14.6).).

ActiveWorkbook.Worksheets.Add \_

Count:=Val(InputBox("Введите количество листов"))

Листинг 14.6. Добавляем в книгу указанное количество листов

Обычно *листы* в *книгах* MS Excel имеют осмысленные имена. Однако, как вы могли заметить, метод Add не предусматривает задание имени вставляемому *листу*, оставляя его в значении по умолчанию. Чтобы переименовать добавленный лист, вам придется пользоваться свойством Name *объекта* *Worksheet*.

**14.9.2. Copy - копируем листы**

[14-06-Excel Worksheets Copy.xlsm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/14/files/14_06.xlsm) - пример к п. 14.9.2.

Позволяет скопировать лист в новое место *книги*.

Принимает два параметра - Before и After, которые, соответственно, позволяют задать положение перемещаемого *листа* до или после какого-то другого *листа*. Например, [листинг 14.7](https://www.intuit.ru/studies/courses/494/350/lecture/8372?page=1#example.14.7). копирует активный лист в позицию после *листа* с именем "Данные".

ActiveWorkbook.ActiveSheet.Copy \_

After:=Worksheets("Данные")

Листинг 14.7. Создаем копию листа в пределах книги

Если метод вызвать без параметров, то будет создана новая *книга*, в которую будет помещена копия *листа*, для которого вызывался этот метод. Например, так, как в [листинге 14.8](https://www.intuit.ru/studies/courses/494/350/lecture/8372?page=1#example.14.8).

ActiveWorkbook.ActiveSheet.Copy

Листинг 14.8. Копируем лист в новую книгу

**14.9.3. Delete, Move - удаляем и перемещаем листы**

[14-07-Excel Worksheets Del Mov.xlsm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/14/files/14_07.xlsm) - пример к п. 14.9.3.

Метод Delete удаляет лист. Например, так мы удаляем первый лист в *коллекции* *Workbooks* ([листинг. 14.9](https://www.intuit.ru/studies/courses/494/350/lecture/8372?page=1#example.14.9).)

ActiveWorkbook.Worksheets.Item(1).Delete

Листинг 14.9. Удаляем лист

Обратите внимание на то, что в этом примере мы обращаемся к *листу* с помощью свойства Item *коллекции* *Worksheets*, которое позволяет работать с *листом*, используя индекс или имя.

Move позволяет переместить лист на новое место в *книге*. В качестве параметров принимает параметры Before и After, которые задают его положение до или после заданного в них *листа*. Например, так ([листинг 14.10](https://www.intuit.ru/studies/courses/494/350/lecture/8372?page=1#example.14.10).) мы перемещаем активный лист в *книге* в позицию, предшествующую *листу* с именем Данные.

ActiveWorkbook.ActiveSheet.Move \_

Before:=Worksheets("Данные")

Листинг 14.10. Перемещаем лист

**14.10. Свойства коллекции Worksheets**

**14.10.1. Count, Item, Visible - работа с объектами**

Мы уже не раз сталкивались с этими свойствами для других объектов. Count позволяет узнать количество объектов в коллекции, Item - обратиться к объекту по имени или индексу, Visible - отображать и скрывать объект.

**14.11. Объект Worksheet**

*Worksheet* - *рабочий лист* - его методы, свойства и события находятся в непосредственной близости от ячеек, с которыми, в основном, ведется работа при программировании для MS *Excel*.

**14.12. Методы объекта Worksheet**

Практически все часто используемые методы Workheet вам уже знакомы - либо по описаниям предыдущих объектов и коллекций *Excel*, либо - по Microsoft *Word*.

**14.12.1. Activate, Calculate, Copy, Delete, Move, Select, PivotTables, ShowAllData**

* Activate - делает лист активным.
* Calculate - пересчитывает лист.
* Copy, Delete, Move - соответственно, копирует, удаляет, перемещает лист в *книге*.
* Select - выделяет лист.
* PivotTables - возвращает коллекцию сводных таблиц, расположенных на *листе*. Подробнее о сводных таблицах мы поговорим в п. 7.10., посвященному работе с базами данных. ShowAllData - показывает все содержимое *рабочего листа* - отменяет скрытие данных при фильтровании и т.д.

**14.12.2. Paste, PasteSpecial, о работе с Selection**

[14-08-Excel Worksheet Paste.xlsm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/14/files/14_08.xlsm) - пример к п. 14.2.2.

Метод Paste вставляет содержимое буфера обмена в указанную область. Для копирования данных в буфер обмена используется метод Copy объекта Range (или объекта Selection ).

PasteSpecial осуществляет специальную вставку. Например, с помощью этого метода можно скопировать содержимое ячеек, содержащих формулы и вставить его в другие ячейки, преобразовав таким образом, чтобы вставлены были лишь вычисленные *значения формул*.

Рассмотрим пример. Внесем в пять горизонтально расположенных ячеек (от A2 до E2 ) какие-нибудь числа, а в шестой ( F2 ) запишем такую формулу: =сумм(A2:E2). В шестой ячейке будет отображена сумма первых пяти. Теперь вручную создадим такой макрос, назвав его DoubleCopy ([листинг 14.11](https://www.intuit.ru/studies/courses/494/350/lecture/8372?page=2#example.14.11).).

Перед запуском макроса выделите область, заполненную данными.

'Копируем выделенную область

'в буфер обмена

Application.Selection.Copy

'Выделяем ячейку A3

Range("A3").Select

'Вставляем содержимое буфера обмена

ActiveSheet.Paste

'Теперь выделяем диапазон ячеек,

'в котором хранятся исходные данные

Range("A2:F2").Select

Application.Selection.Copy

Range("A4").Select

'Вставляем значения - без формул

Selection.PasteSpecial Paste:=xlPasteValues

Листинг 14.11. Работа с буфером обмена

Смысл команд этой программы ясен из комментариев. Обратите внимание на то, что здесь мы работали с объектом Selection. Так же посмотрите на содержимое ячеек в файле-примере. При вставке скопированного методом Paste, вставляется абсолютно все - в том числе и формулы, а при использовании PasteSpecial со свойством Paste, установленным в xlPasteSpecial, вставляются лишь значения, без формул.

**14.12.3. Protect и Unprotect - защита и снятие защиты**

Метод Protect защищает лист от изменений. Его можно вызывать без параметров, однако, при необходимости вы можете очень тонко настроить защиту *листа*. Например, задать пароль, разрешить пользователю некоторые операции и т.д.

Метод *Unprotect* снимает защиту с *листа*.

Установка защиты, особенно - защиты с паролем - может быть полезной, если вы хотите, чтобы пользователь гарантированно не мог внести изменения в какой-либо из *листов* вашей *книги*.

**14.13. Свойства объекта Worksheet**

**14.13.1. Cells - ячейки листа**

Это свойство возвращает объект типа Range, который представляет собой все ячейки *листа*. С помощью Cells можно обращаться к каждой конкретной ячейке по отдельности. Например, чтобы записать число 8 в ячейку, находящуюся на пересечении первой строки и пятого столбца текущего *листа*, надо выполнить такую команду ([листинг 14.12](https://www.intuit.ru/studies/courses/494/350/lecture/8372?page=2#example.14.12).)

ActiveSheet.Cells(1, 5) = 8

Листинг 14.12. Вставка данных в ячейку, используя стиль R1C1

Выше мы пользовались свойством Cell для того, чтобы работать с данными на *листе*. Помимо записи данных в ячейки, данные можно считывать, ячейки можно форматировать и т.д. Об этом - ниже.

**14.13.2. EnableCalculation - управление автоматическим пересчетом ячеек**

Позволяет включать и отключать автоматический пересчет ячеек. В проектах большого размера пересчет ячеек может занимать много системных ресурсов.

Это свойство можно установить в False если вы, например, осуществляете вывод большого объема данных на лист и хотите ускорить этот процесс.

**14.13.3. Next и Previous - перемещение по листам книги**

Эти свойства позволяют получать ссылки на следующий, и, соответственно, предыдущий лист *книги*.

**14.13.4. QueryTables - таблицы запросов**

Возвращает коллекцию QueryTables, содержащую объекты QueryTable - таблицы запросов, которые используются для вставки в документ информации из баз данных.

**14.13.5. Range - работа с ячейками и диапазонами ячеек**

Это свойство возвращает объект Range, соответствующей ячейке или *диапазону ячеек*, указанных при его вызове. Собственно говоря, Range наряду с Cells - это основные инструменты для работы с ячейками *листа*. Например, так ([листинг 14.13](https://www.intuit.ru/studies/courses/494/350/lecture/8372?page=2#example.14.13).) можно вставить число 4 в ячейку E1:

ActiveSheet.Range("E1") = 4

Листинг 14.13. Добавляем данные в ячейку

А такая конструкция ([листинг 14.14](https://www.intuit.ru/studies/courses/494/350/lecture/8372?page=2#example.14.14).) позволяет прибавить по 1 к значению каждой из ячеек диапазона A1:K100

Dim MyCell As Variant

For Each MyCell In ActiveSheet.Range("A1:K100")

MyCell.Value = MyCell.Value + 1

Next

Листинг 14.14. Работа с диапазоном ячеек

При использовании конструкции For-Each обход ячеек осуществляется по строкам.

Ниже мы рассмотрим объект Range подробнее.

**14.13.6. UsedRange - заполненные ячейки**

Это свойство возвращает объект Range, содержащий все заполненные ячейки на *листе*. Это может быть полезным, например, если вы хотите скопировать эти ячейки на другой лист и в других подобных ситуациях.

**14.13.7. Visible - скрытие и отображение листов**

Позволяет скрывать и отображать *листы*. По умолчанию *листы* отображаются - их свойство Visible установлено в True, установив его в False, вы скрываете лист.

**14.14. События объекта Worksheet**

[14-09-Excel Worksheet Change.xlsm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/14/files/14_09.xlsm) - пример к п. 14.14.

Работа с событиями *листов* очень похожа на работу с событиями книг. Вы могли заметить, что в VBA-проекте, присоединенном к *книге*, есть объекты каждого из добавленных в *книгу* *листов*. Для создания обработчика события *листа* достаточно сделать *двойной щелчок* по его объекту и в появившемся окне редактора выбрать *объект* (для каждого *листа* это будет *Worksheet* ), и, как обычно, выбрать событие. *Лист* может воспринимать следующие события: Activate, BeforeDoubleClick, BeforeRightClick, Calculate, Change, Deactivate, FollowHyperlink, PivotTableUpdate, SelectionChange. Эти события практически полностью идентичны аналогичным событиям для *объекта Workbook*. Главное отличие заключается в том, что они имеют смысл для каждого конкретного *листа*, а не для всех *листов*, как в случае с *книгой*.

Используем событие Change для проверки данных, вводимых пользователем в различные ячейки *рабочего листа*. Например, давайте напишем программу, которая предлагает пользователю создать *пароль* и тут же сообщает о том, насколько его *пароль* надежен. В качестве критерия надежности будем использовать длину пароля:

* Пароль высокой надежности: длина не менее 8 символов
* Пароль средней надежности: длина 5-7 символов
* Пароль низкой надежности: до 5 символов.

Когда *пользователь* ввел *пароль* в ячейку, в соседней ячейке должен появляться прогноз надежности пароля. Пусть *пользователь* будет вводить *пароль* в ячейку A2, в ячейке B2 будет отображаться прогноз.

Для решения этой задачи создадим обработчик события Change для *листа*, на котором будут расположены вышеуказанные поля. Добавим в обработчик такой код ([листинг 14.15](https://www.intuit.ru/studies/courses/494/350/lecture/8372?page=2#example.14.15).)

'Для хранения пароля

Dim str\_Pass As String

str\_Pass = ActiveSheet.Range("A2")

If Len(str\_Pass) >= 8 Then \_

ActiveSheet.Range("B2") = "Надежный пароль"

If Len(str\_Pass) < 8 And Len(str\_Pass) >= 5 Then \_

ActiveSheet.Range("B2") = \_

"Пароль средней надежности"

If Len(str\_Pass) < 5 Then \_

ActiveSheet.Range("B2") = \_

"Ненадежный пароль"

Листинг 14.15. Проверка данных при вводе в ячейку

**14.15. Выводы**

В этой лекции мы познакомились с *книгами* и *листами* MS *Excel*. В следующей лекции мы займемся работой с ячейками *листа*.

Лекция 15. Работа с ячейками - объект Range

**15.1. Как обратиться к ячейке**

[15-01-Excel Обращение к ячейкам.xlsm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/15/files/15_01.xlsm) - пример к п. 15.1.

Мы добрались до ячеек, работа с которыми осуществляется, в основном, через *объект* типа Range. Выше мы немного работали с *ячейками*, а теперь рассмотрим их наиболее интересные методы и свойства.

Выше мы уже обращались к *ячейкам* в некоторых примерах. Здесь мы кратко обобщим и поясним основные способы обращения к *ячейкам*.

Можно адресовать *ячейку* или *диапазон ячеек*, указав их адреса в стиле A1. Здесь и далее мы используем метод Select *объекта* Range, который выделяет *ячейки* ([листинг 15.1](https://www.intuit.ru/studies/courses/494/350/lecture/8373?page=1#example.15.1).)

ActiveSheet.Range("A2").Select

Листинг 15.1. Обращаемся к ячейке по имени в стиле A1

Для обращения к диапазону ячеек нужно знать верхнюю левую и нижнюю правую границы диапазона. Например, для обращения к диапазону высотой в одну строку от A2 до E2 или к диапазону A2:E4 - понадобится такой код ([листинг 15.2](https://www.intuit.ru/studies/courses/494/350/lecture/8373?page=1#example.15.2).)

ActiveSheet.Range("A2:E2").Select

ActiveSheet.Range("A2:E4").Select

Листинг 15.2. Обращаемся к диапазонам

Можно воспользоваться конструкцией с использованием объекта Cells, который позволяет обращаться к отдельной *ячейке* по ее индексу в формате R1C1. Чтобы обратиться к *ячейке* A5 таким способом, нужно заметить, что она расположена в пятой строке и первом столбце ([листинг 15.3](https://www.intuit.ru/studies/courses/494/350/lecture/8373?page=1#example.15.3).):

ActiveSheet.Cells(5,1).Select

Листинг 15.3. Обращаемся к ячейке по номеру строки и столбца

Можно объединить использование Range и Cells, указав *координаты* ячеек при адресации диапазона с помощью Cells ([листинг 15.4](https://www.intuit.ru/studies/courses/494/350/lecture/8373?page=1#example.15.4).).

ActiveSheet.Range(Cells(5, 4), \_

Cells(7, 5)).Select

Листинг 15.4. Обращение к диапазону с использованием комбинации Range и Cells

Нам уже встречалось использование Cells для доступа к группам ячеек в цикле - в качестве индексов ячеек можно использовать переменные ([листинг 15.5](https://www.intuit.ru/studies/courses/494/350/lecture/8373?page=1#example.15.5).)

For i = 1 To 3

For j = 1 To 3

ActiveSheet.Cells(i, j).Select

Application.Wait (Now + \_

TimeValue("0:00:01"))

p = p + 1

Selection = p

Next j

Next i

ActiveSheet.Range("A1:E5").Clear

Листинг 15.5. Обращение к ячейкам в цикле

Здесь мы циклически выделяем *ячейки* диапазона A1:C3, делая задержку на 1 секунду после каждого выделения и выводя количество прошедших с начала работы программы секунд. Здесь мы воспользовались для выделения *ячейки* уже знакомым вам методом Select, а для ввода данных в выделенную *ячейку* применили *объект* Selection, который в данном случае ссылается на выделенную *ячейку*. В конце мы очистили *диапазон* A1:E5 от введенных данных.

*Объект* Selection - это еще один способ работы с *ячейками*, однако он используется сравнительно редко, так как к *ячейкам* удобнее обращаться по их именам.

Выше мы использовали прямое обращение к *ячейкам* активного листа, без использования *объектных переменных*.)

Dim obj\_MyCells As Range

Set obj\_MyCells = ActiveSheet.Cells(5, 5)

obj\_MyCells.Select

Листинг 15.6. Объектная переменная и работа с ячейками

Помимо обращения к отдельным *ячейкам* или их диапазонам, *Excel* предусматривает возможность обращения к строкам и столбцам, а так же - к листу целиком.

В [листинге 15.7](https://www.intuit.ru/studies/courses/494/350/lecture/8373?page=1#example.15.7) мы сначала выделяем столбец A, потом столбец B, используя коллекцию Columns (столбцы), 3-ю строку, используя коллекцию Rows (строки) а далее - *лист* целиком.

ActiveSheet.Range("A:A").Select

ActiveSheet.Columns("B:B").Select

ActiveSheet.Range("3:3").Select

ActiveSheet.Rows("4:4").Select

ActiveSheet.Cells.Select

Листинг 15.7. Работа со столбцами, строками и всеми ячейками листа

Еще один способ обращения к *ячейкам* - применение именованных диапазонов (коллекция Names ) мы рассмотрим ниже. А теперь поговорим о методах и свойствах *объекта* Range.

**15.2. Методы Range**

**15.2.1. Activate - активация ячейки**

[15-02-Range Activate.xlsm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/15/files/15_02.xlsm) - пример к п. 15.2.1.

Позволяет выбрать *ячейку* в выделенном диапазоне. Даже когда выделен *диапазон ячеек*, активной является лишь одна из них. Чтобы изменить эту активную *ячейку*, и применяется данный метод. Если использовать вместо метода Activate метод Select, то *ячейка* будет выделена, а остальное выделение - снято. В то же время, если попытаться активировать *ячейку*, расположенную вне выделенного диапазона, выделение снимется, и активированная *ячейка* окажется выделенной.

Например, в [листинге 15.8](https://www.intuit.ru/studies/courses/494/350/lecture/8373?page=1#example.15.8). мы сначала выделили *диапазон ячеек*, а потом, не снимая выделения, сделали одну из ячеек диапазона активной.

Range("A1:E5").Select

Range("C2").Activate

Листинг 15.8. Активация ячейки в выделенной области

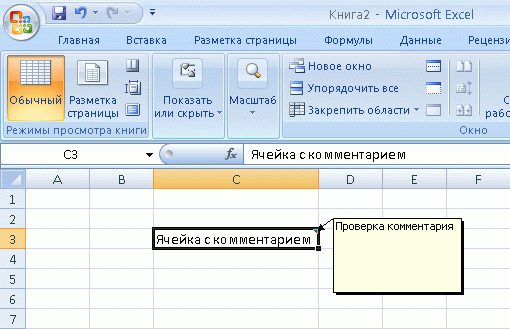
**15.2.2. AddComment - добавляем комментарии к ячейкам**

Позволяет добавлять комментарии к *ячейкам*. Если вы формируете какой-нибудь Excel-документ программно, вы можете добавить в некоторые *ячейки* комментарии для пояснения данных, которые в них хранятся. В [листинге 15.9](https://www.intuit.ru/studies/courses/494/350/lecture/8373?page=1#example.15.9). мы добавляем комментарий к *ячейке* C3.

Range("C3").AddComment ("Проверка комментария")

Листинг 15.9. Добавляем комментарий к ячейке

В правом верхнем углу *ячейки* появится красный треугольник, а наведя мышь на *ячейку*, можно увидеть текст комментария ([рис. 15.1](https://www.intuit.ru/studies/courses/494/350/lecture/8373?page=1#image.15.1).).



**Рис. 15.1.**Комментарий в ячейке MS Excel

**15.2.3. AutoFit - автонастройка ширины столбцов и высоты строк**

Позволяет автоматически подстроить ширину столбцов и высоту строк, входящих в диапазон. Это удобно делать, чтобы придать автоматически генерируемым таблицам привлекательный вид.

Метод можно применять как к диапазону, так и к отдельным строкам или столбцам.

Например, код в [листинге 15.10](https://www.intuit.ru/studies/courses/494/350/lecture/8373?page=1#example.15.10). позволяет автоматически подобрать ширину столбцов A, B, C, D, E, руководствуясь данными, расположенными в первой строке этих столбцов. Если в других строках столбцов будут более длинные значения - они не будут приняты во внимание.

ActiveSheet.Range("A1:E1").Columns.AutoFit

Листинг 15.10. Автоматически настраиваем ширину столбцов по ширине данных в указанных ячейках

Мы не случайно обращаемся здесь к свойству Columns *объекта* Range - иначе метод AutoFit не работает. Если же в подобном вызове не задавать конкретной строки, а выполнить эту команду так ([листинг 15.11](https://www.intuit.ru/studies/courses/494/350/lecture/8373?page=1#example.15.11).), то ширина столбцов A - E будет подстроена таким образом, чтобы наилучшим образом вместить самое длинное из значений, хранящихся в *ячейках*, принадлежащих столбцам.

ActiveSheet.Range("A:E").Columns.AutoFit

Листинг 15.11. Автоматически настраиваем ширину столбцов

**15.2.4. Clear, ClearComments, ClearContents, ClearFormats - очистка и удаление**

Метод Clear позволяет очистить диапазон - он удаляет данные и форматирование из ячеек. Например, в [листинге 15.12](https://www.intuit.ru/studies/courses/494/350/lecture/8373?page=1#example.15.12). мы очищаем от форматирования сначала диапазон A1:E5, а потом - весь лист.

ActiveSheet.Range("A1:E5").Clear

Activesheet.Cells.Select

Selection.Clear

Листинг 15.12. Очистка от данных и форматирования

Другие методы, название которых начинается с Clear, позволяют очищать *ячейки* от соответствующих им объектов.

ClearContents очищает содержимое ячеек, не затрагивая форматирование. Если вы выделите *ячейки* и нажмете клавишу Del на клавиатуре - вы добъетесь того же эффекта.

ClearFormats очищает лишь форматирование ячеек, не затрагивая содержимого.

**15.2.5. Copy, Cut, PasteSpecial - буфер обмена**

Выше мы уже рассматривали команды для работы с буфером обмена в *MS Excel*. Метод Copy копирует содержимое диапазона в буфер обмена, Cut - вырезает, PasteSpecial осуществляет специальную вставку.

Как ни странно, *объект* Range не поддерживает метод Paste, осуществляющий обычную вставку, однако, этот метод поддерживает объект *Worksheet*.

**15.2.6. Delete - удалить диапазон**

Удаляет выделенный диапазон - остальные *ячейки* сдвигаются, занимая его место.

**15.2.7. Merge, UnMerge - объединение ячеек**

[15-03-Range Merge.xlsm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/15/files/15_03.xlsm) - пример к п. 15.2.7.

Merge позволяет создать одну объединенную *ячейку* из заданного диапазона.

UnMerge разбивает объединенную *ячейку* на обычные *ячейки*.

Объединенные *ячейки* удобно использовать для хранения в них названий таблиц.

В [листинге 15.13](https://www.intuit.ru/studies/courses/494/350/lecture/8373?page=1#example.15.13). мы программно формируем таблицу шириной в 10 ячеек. Заполняем ее данными, автоматически подстраиваем ширину столбцов под введенные значения. После этого вводим в левую верхнюю *ячейку* строки, которая расположена над таблицей, название таблицы, и объединяем все *ячейки* до конца таблицы, расположенные левее строки с названием. В итоге название будет отображено в одной большой строчке, занимающей всю верхнюю часть таблицы ([рис. 15.2](https://www.intuit.ru/studies/courses/494/350/lecture/8373?page=1#image.15.2).).

'Заполняем область C3:L2

'случайными целыми числами

For i = 1 To 10

For j = 1 To 10

ActiveSheet.Cells(i + 2, j + 2) = \_

Int(Rnd \* 100)

Next j

Next i

'Выравниваем размер столбцов

ActiveSheet.Range("C:L").Columns.AutoFit

'Записываем название таблицы

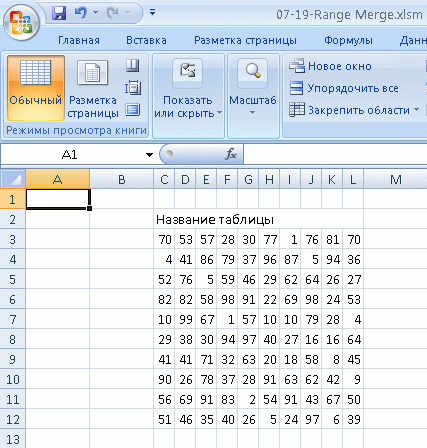
'в ячейку верхней строчки

Range("C2") = "Название таблицы"

'Объединяем ячейки над таблицей

Range("C2:L2").Merge

Листинг 15.13. Оформление таблиц с использованием объединения ячеек



**Рис. 15.2.**Название таблицы в объединенной ячейке

**15.2.8. Select - выделение ячейки**

[15-04-Range Select.xlsm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/15/files/15_04.xlsm) — пример к п. 7.7.2.8.

Выделяет *ячейки* или *ячейку*. Выделив *ячейку*, к ней можно обращаться, используя объект Selection. Так же этот объект можно использовать для работы с *ячейками*, предварительно выделенными пользователями.

Например, в [листинге 15.14](https://www.intuit.ru/studies/courses/494/350/lecture/8373?page=1#example.15.14). мы находим сумму чисел, которые хранятся в *ячейках* диапазона, выделенного пользователем перед запуском макроса.

Dim obj\_Range As Range

Dim num\_Sum

'Обращаемся к каждой ячейке

'в выделенной области

For Each obj\_Range In Selection.Cells

num\_Sum = num\_Sum + Val(obj\_Range)

Next

MsgBox ("Сумма выделенных ячеек: " & \_

num\_Sum)

Листинг 15.14. Поиск суммы чисел в диапазоне, выделенном пользователем

**15.3. Свойства Range**

**15.3.1. Address - адрес ячейки в формате A1**

[15-05-Range Address.xlsm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/15/files/15_05.xlsm) - пример к п. 15.3.1.

Возвращает строку, представляющую собой адрес *ячейки* в формате A1. Адрес выводится в абсолютном виде - снабжается знаками $.

[Листинг 15.15](https://www.intuit.ru/studies/courses/494/350/lecture/8373?page=2#example.15.15) позволяет, задав адрес ячейки в виде R1C1, вывести ее адрес в формате A1.

Dim num\_Row

Dim num\_Col

Dim MyRange As Range

num\_Row = Val(InputBox("Введите строк"))

num\_Col = Val(InputBox("Введите столбец"))

Set MyRange = \_

ActiveSheet.Cells(num\_Row, num\_Col)

MsgBox (MyRange.Address + \_

" - имя ячейки " & \_

" с индексами " & num\_Row & " и " & num\_Col)

Листинг 15.15. Вывод адреса ячейки в формате A1

**15.3.2. Areas - работа с несмежными выделенными областями**

[15-06-Range Areas.xlsm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/15/files/15_06.xlsm) - пример к п. 15.3.2.

Свойство возвращает коллекцию Areas, которая содержит все *объекты* типа Range в выделенной области в том случае, если выделенная область содержит несмежные диапазоны ячеек. Эту коллекцию удобно использовать для обработки несмежных областей, выделенных пользователем. Например, [листинг 15.16](https://www.intuit.ru/studies/courses/494/350/lecture/8373?page=2#example.15.16). выводит каждую из выделенных областей большой таблицы в отдельный лист.

'Для хранения исходной

'выделенной области

Dim obj\_Area As Range

'Для хранения ссылки на

'отдельные диапазоны

Dim obj\_Range As Range

'Для исходного листа

Dim obj\_Sheet As Worksheet

'Для каждого из новых листов

Dim obj\_OldSheet As Worksheet

'Присвоим ссылку на выделенную область

Set obj\_Area = Selection

'Ссылка на активный лист

Set obj\_OldSheet = ActiveSheet

'Для каждой несмежной области в

'выделении

For Each obj\_Range In obj\_Area.Areas

'Копируем эту область

obj\_OldSheet.Activate

obj\_Range.Select

Selection.Copy

'Создаем новый лист

'и вставляем в него

Set obj\_Sheet = Worksheets.Add

obj\_Sheet.Select

obj\_Sheet.Paste

Next

Листинг 15.16. Вывод выделенных несмежных областей в отдельные листы

**15.3.3. Borders - управление границами ячеек**

[15-07-Range Borders.xlsm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/15/files/15-07.xlsm) - пример к п. 15.3.3.

Позволяет управлять границами ячеек. Границы ячеек обычно используются для оформления таблиц. Как правило, работа ведется с неким выделенным *диапазоном ячеек*, для которого настраивают внешние границы, внутренние границы, типы и цвета линий. Работа с границами ячеек ведется посредством свойств объектов Border. Собственно говоря, самое важное свойство коллекции Borders - это Item, дающее доступ к отдельным объектам Border - то есть к границам. Остальные действия с границами проводятся с помощью свойств объектов Border.

Item - свойство, которое возвращает отдельную границу по индексу или имени границы. Принято использовать следующие имена границ:

* xlDiagonalDown - диагональ из левого верхнего угла *ячейки* в правый нижний
* xlDiagonalUp - диагональ из левого нижнего угла *ячейки* в правый верхний
* xlEdgeBottom - нижняя внешняя граница
* xlEdgeLeft - левая внешняя граница
* xlEdgeRight - правая внешняя граница
* xlEdgeTop - верхняя внешняя граница
* xlInsideHorizontal - внутренние горизонтальные границы
* xlInsideVertical - внутренние вертикальные границы

Когда выбрана граница, с которой вы хотите работать, можно использовать свойства объекта Border, в частности, следующие:

Color - позволяет задавать цвет границы. Для задания цвета можно использовать функцию RGB, которая по переданным ей значениям цветовых компонентов в формате RGB возвращает нужный цвет. Например, такой вызов этой функции возвратит красный цвет: RGB(255,0,0). Также здесь можно использовать цветовые константы: vbBlack, vbRed и т.д.

LineStyle - позволяет задавать тип линии. Здесь применимо несколько констант. В частности, следующие:

* xlContinuous - непрерывная линия
* xlDash - линия, состоящая из черточек
* xlDashDot - линия с чередующимися точками и черточками
* xlDot - линия состоящая из точек
* xlDouble - двойная линия
* xlLineStyleNone - нет линий

Weight - задает толщину линии при помощи указания одной из констант:

* xlHairline - самая тонкая линия
* xlThin - тонкая линия
* xlMedium - линия средней толщины
* xlThick - толстая линия

Давайте рассмотрим пример ([листинг 15.17](https://www.intuit.ru/studies/courses/494/350/lecture/8373?page=2#example.15.17).). Выведем набор значений в таблицу, отформатируем ее таким образом, чтобы внешние границы состояли из сплошных черных линий средней толщины, внутренние - из точечных тонких красных линий ([рис. 15.3](https://www.intuit.ru/studies/courses/494/350/lecture/8373?page=2#image.15.3).)

Dim obj\_Range As Range

'Добавляем в книгу новый лист

'он автоматически становится активным

Worksheets.Add

ActiveSheet.Name = "Новая таблица"

'заполняем небольшую таблицу данными

For i = 1 To 5

For j = 1 To 5

ActiveSheet.Cells(i + 1, j + 1) = \_

Int(Rnd \* 100)

Next j

Next i

'Свойство CurrentRegion возвращает

'заполненную данными область вокруг

'ячейки, для которой вызывается

Set obj\_Range = ActiveSheet.Cells(i, j).CurrentRegion

'Настраиваем свойства каждой из границ

With obj\_Range.Borders(xlEdgeLeft)

.LineStyle = xlDContinuous

.Color = vbBlack

.Weight = xlMedium

End With

With obj\_Range.Borders(xlEdgeTop)

.LineStyle = xlContinuous

.Color = vbBlack

.Weight = xlMedium

End With

With obj\_Range.Borders(xlEdgeBottom)

.LineStyle = xlContinuous

.Color = vbBlack

.Weight = xlMedium

End With

With obj\_Range.Borders(xlEdgeRight)

.LineStyle = xlContinuous

.Color = vbBlack

.Weight = xlMedium

End With

With obj\_Range.Borders(xlInsideVertical)

.LineStyle = xlDot

.Color = RGB(255, 0, 0)

.Weight = xlThin

End With

With obj\_Range.Borders(xlInsideHorizontal)

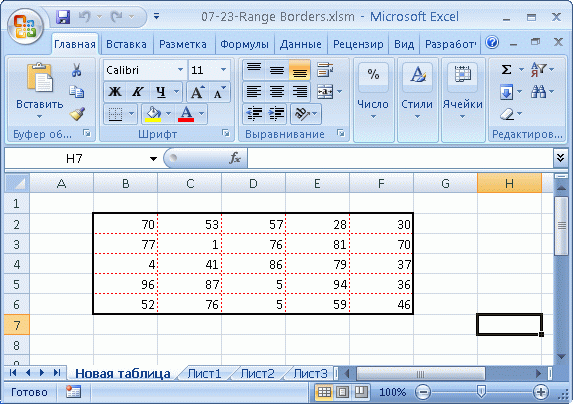
.LineStyle = xlDot

.Color = RGB(255, 0, 0)

.Weight = xlThin

End With

Листинг 15.17. Форматирование границ



**Рис. 15.3.**Отформатированная таблица в документе

**15.3.4. Cells, Columns, Rows - ячейки, столбцы, строки**

[15-08-Range Cells.xlsm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/15/files/15_08.xlsm) - пример к п. 15.3.4.

Свойство Cells позволяет обращаться к отдельным *ячейкам* в диапазоне. При работе с этим свойством в отдельном *диапазоне ячеек* нумерация ячеек ведется по собственной системе координат. Иными словами, диапазон выступает как небольшой виртуальный рабочий лист: левая верхняя *ячейка* диапазона получает индекс (1, 1), *ячейка*, расположенная во втором столбце и третьей строке диапазона, - индекс (3,2) и т.д.

Свойства Columns и Rows возвращают, соответственно, коллекции, которые содержат столбцы и строки диапазона. Например, так ([листинг 15.18](https://www.intuit.ru/studies/courses/494/350/lecture/8373?page=3#example.15.18).) можно узнать количество строк в диапазоне, ссылка на который хранится в переменной obj\_Range.

num\_Rows = obj\_Range.Rows.Count

Листинг 15.18. Количество строк в диапазоне

Воспользуемся свойством Cell для диапазона размером 6х5 ячеек, чтобы заполнить этот диапазон данными ([листинг 15.19](https://www.intuit.ru/studies/courses/494/350/lecture/8373?page=3#example.15.19).).

Dim obj\_Range As Range

Set obj\_Range = ActiveSheet.Range("B2:F7")

For i = 1 To obj\_Range.Rows.Count

For j = 1 To obj\_Range.Columns.Count

obj\_Range.Cells(i, j) = \_

Int(Rnd \* 100)

Next j

Next i

Листинг 15.19. Заполнение диапазона ячеек с использованием свойства Cell

Как видите, мы присваиваем ссылку на *диапазон ячеек* B2:F7 переменной obj\_Range, после чего в цикле, используя свойство Cells для этой переменной, заполняем выбранный диапазон значениями. Еще раз обращаю ваше внимание на то, что свойство Cells для Range "работает" в пределах диапазона и *ячейки* в диапазоне имеют собственную нумерацию, отличную от ячеек листа

**15.3.5. CurrentRegion - область, заполненная данными**

Возвращает *объект* Range, который представляет собой все заполненные данными *ячейки* вокруг *ячейки* или диапазона, для которого вызывается.

**15.3.6. Characters, Font - форматирование текста**

[15-09-Range Font.xlsm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/15/files/15_09.xlsm) - пример к п. 15.3.6.

Свойство Characters позволяет настраивать форматирование отдельных символов текста *ячейки*, а параметр Font нужен для настройки параметров шрифта по *ячейке* в целом. Рассмотрим пример. В *ячейке*, выделенной пользователем, отформатируем текст шрифтом Times New Roman, размером 15, красного цвета. А первый символ отформатируем курсивом ([листинг 15.20](https://www.intuit.ru/studies/courses/494/350/lecture/8373?page=3#example.15.20).).

Dim obj\_Range As Range

Set obj\_Range = Selection

With obj\_Range

.Font.Name = "Times New Roman"

.Font.Size = 15

.Font.Color = vbRed

.Characters(1, 1).Font.Italic = True

End With

Листинг 15.20. Форматирование текста в ячейке

Свойство Name объекта Font позволяет задать имя шрифта, Size - размер, Color - цвет. При работе с Characters мы задаем номер символа, с которого начинается форматирование, а так же - количество символов, начиная с первого. Чтобы отформатировать второй и третий символы, нам понадобилось бы вызвать это свойство так: Characters(2, 2) - два символа, начиная со второго.

**15.3.7. Formula, FormulaR1C1 - формулы в ячейках**

[15-10-Range Formula.xlsm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/15/files/15_10.xlsm) - пример к п. 15.3.7.

*Formula* позволяет записать в *ячейку* формулу, а также - узнать, какая формула записана в *ячейке*. Формулы используют ссылки на *ячейки* в стиле A1. Например, для записи в *ячейку* A1 суммы ячеек A2 и A3, нам понадобится такая команда ([листинг 15.21](https://www.intuit.ru/studies/courses/494/350/lecture/8373?page=3#example.15.21).)

Range("A1").Formula = "=$A$2+$A$3"

Листинг 15.21. Ввод формулы в ячейку

Свойство FormulaR1C1 записывает в *ячейку* формулу, используя стиль ссылок R1C1.

Рассмотрим пример. Программно создадим таблицу с числами и рассчитаем после каждого столбца и каждой строки таблицы суммы составляющих их элементов. Для расчетов сумм строк используем свойство *Formula*, для расчета сумм по столбцам - FormulaR1C1 ([листинг 15.22](https://www.intuit.ru/studies/courses/494/350/lecture/8373?page=3#example.15.22).).

'Для хранения ссылки на ячейку

'в которую запишем формулу

Dim obj\_Range As Range

'Для хранения ссылки на первую

'ячейку диапазона (для формулы в стиле A1)

Dim obj\_Range1 As Range

'Для хранения ссылки на последнюю

'ячейку диапазона (для формулы в стиле A1)

Dim obj\_Range2 As Range

'Для сборки адресов ячеек в

'стиле R1C1

Dim str\_R1C11 As String

Dim str\_R1C12 As String

'Заполняем ячейки данными

For i = 1 To 5

For j = 1 To 5

ActiveSheet.Cells(i, j) = \_

Int(Rnd \* 100)

Next j

Next i

'Заполняем формулами ячейки, которые

'будут отображать суммы по строкам

For i = 1 To 5

'Ссылка на ячейку с формулой

Set obj\_Range = ActiveSheet.Cells(i, 6)

'Ссылка на первую ячейку диапазона

Set obj\_Range1 = ActiveSheet.Cells(i, 1)

'На последнюю ячейку

Set obj\_Range2 = ActiveSheet.Cells(i, 5)

'Формула передается в ячейку в виде строки

'формируем строку такого вида:

'=SUM($A$1:$A$5)

obj\_Range.Formula = \_

"=sum(" + obj\_Range1.Address + ":" + \_

obj\_Range2.Address + ")"

Next i

'Заполняем ячейки с суммами по

'столбцам

For i = 1 To 5

'ссылка на ячейку с формулой

Set obj\_Range = ActiveSheet.Cells(6, i)

'Собираем ссылку на первую ячейку

'она будет иметь вид R1C1 для первого

'столбца, R1C2 для второго и т.д.

str\_R1C11 = "R" + Mid(Str(1), 2, \_

Len(Str(1)) - 1) + \_

"C" + Mid(Str(i), 2, Len(Str(i)) - 1)

'Собираем ссылку на последнюю ячейку

'она будет иметь вид R5C1 для первого

'столбца, R5C2 для второго и т.д.

str\_R1C12 = "R" + Mid(Str(5), 2, \_

Len(Str(5)) - 1) + \_

"C" + Mid(Str(i), 2, Len(Str(i)) - 1)

'Записываем в ячейку формулу вид

obj\_Range.FormulaR1C1 = \_

"=sum(" + str\_R1C11 + ":" + \_

str\_R1C12 + ")"

Next i

Листинг 15.22. Формирование таблицы с вводом формул подсчета итогов по строкам и столбцам

**15.3.8. Interior - внешний вид ячейки**

Позволяет управлять внешним видом *ячейки*. В частности, настраивать ее цвет.

Например, такой код ([листинг 15.23](https://www.intuit.ru/studies/courses/494/350/lecture/8373?page=3#example.15.23).) позволяет окрасить *ячейку* A1 в красный цвет

Range("A1").Interior.Color = vbRed

Листинг 15.23. Окраска ячейки в красный цвет

**15.3.9. Name - работа с именованными диапазонами**

[15-11-Range Name.xlsm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/15/files/15_11.xlsm) - пример к п. 15.3.9.

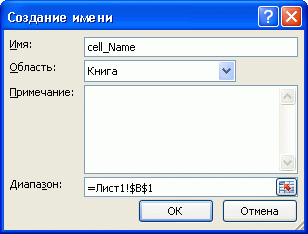
Позволяет узнать или установить имя *ячейки* или *диапазона ячеек*. Имена организованы в коллекцию Names. Именами удобно пользоваться для автоматического заполнения каких-либо заранее созданных таблиц.

Рассмотрим пример. Создадим на рабочем листе *MS Excel* таблицу такого вида ([табл. 15.1](https://www.intuit.ru/studies/courses/494/350/lecture/8373?page=3#table.15.1).)

|  |  |
| --- | --- |
| Таблица 15.1. Структура таблицы на листе *MS Excel* | |
| **Имя** | **(сюда будет введено имя)** |
| Возраст | (сюда будет введен возраст) |

Дадим *ячейкам*, в которые должны быть введены имя и возраст, имена - cell\_Name и cell\_Age.

Для задания имени щелкните правой кнопкой мыши по *ячейке* и выберите в появившемся меню команду Имя диапазона ([рис. 15.4](https://www.intuit.ru/studies/courses/494/350/lecture/8373?page=3#image.15.4).)



**Рис. 15.4.**Задаем имя ячейке

Заполните поле Имя, здесь же можно выбрать область видимости имени, ввести примечание и отредактировать, при необходимости, диапазон, для которого назначается имя.

Также для присвоения имени *ячейке* или диапазону вы можете воспользоваться полем, расположенным слева от строки формулы - в этом поле отображается имя активной *ячейки*. Просто впишите туда нужное имя.

Теперь добавим на лист кнопку и присвоим ее обработчику Click такой код ([листинг 15.24](https://www.intuit.ru/studies/courses/494/350/lecture/8373?page=3#example.15.24).).

'Переменная для работы с ячейкой

Dim obj\_Range As Range

'Переменная для адреса ячейки

Dim str\_Name As String

'Так как имя ячейки возвращается

'в виде =лист1$A$1 мы вырезаем из

'переданного значения все, кроме знака =

str\_Name = Mid( \_

ActiveWorkbook.Names("cell\_Name"), \_

2, Len(ActiveWorkbook.Names("cell\_Name")) - 1)

'Вводим в ячейку данные

Range(str\_Name).Value = \_

InputBox("Введите имя")

str\_Name = Mid( \_

ActiveWorkbook.Names("cell\_Age"), \_

2, Len(ActiveWorkbook.Names("cell\_Age")) - 1)

Range(str\_Name).Value = \_

Val(InputBox("Введите возраст"))

Листинг 15.24. Работа с именованными ячейками

Здесь мы используем коллекцию Names объекта ActiveWorkbook, чтобы обратиться к конкретной *ячейке*. Здесь мы задавали имена ячеек вручную, но используя свойство Name можно задавать их в автоматическом режиме. Например, для присвоения *ячейке* A1 имени cell\_A, надо выполнить такой код ([листинг 15.25](https://www.intuit.ru/studies/courses/494/350/lecture/8373?page=3#example.15.25).)

Range("A1").Name = "cell\_A"

Листинг 15.25. Программная установка имени ячейки

По такой же схеме осуществляется работа с именованными диапазонами ячеек.

**15.3.10. Value - содержимое ячейки**

[15-12-Range Value.xlsm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/15/files/15_12.xlsm) - пример к п. 15.3.10.

Позволяет узнать или установить содержимое *ячейки*. Того же эффекта можно добиться, если обращаться к *ячейке* без указания каких-либо свойств.

Рассмотрим пример - здесь мы получаем с помощью свойства Value значение, хранящееся в *ячейке*, если оно меньше 0 - записываем в *ячейку* модуль хранящегося в ней числа и меняем цвет *ячейки* на vbCyan (голубой) ([листинг 15.25](https://www.intuit.ru/studies/courses/494/350/lecture/8373?page=3#example.15.25).)

Dim obj\_Cell As Range

For Each obj\_Cell In ActiveSheet.Range("A1:E8")

If obj\_Cell.Value < 0 Then

obj\_Cell.Value = Abs(obj\_Cell.Value)

obj\_Cell.Interior.Color = vbCyan

End If

Next

Листинг 15.25. Работа с содержимым ячейки

**15.4. Выводы**

В этой лекции мы обсудили особенности работы с *объектом* Range, который предоставляет средства взаимодействия с *ячейками* рабочих листов Microsoft *Excel*. В следующей лекции мы рассмотрим некоторые дополнительные приемы работы с *MS Excel*.

Лекция 16. Дополнительные сведения о программировании для MS Excel

**16.1. Вычисления и формулы**

[16-01-Formula.xlsm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/16/files/16_01.xlsm) - пример к п. 16.1.

Как вы знаете, *MS Excel* поддерживает огромное количество формул. Однако, с их использованием в *VBA* есть одна небольшая сложность. В локализованной версии *VBA*, в частности, в русскоязычной, *формулы*, которые отображаются в ячейках, имеют русскоязычное написание. Например, такая *формула*: =сумм(A1:*A10*) посчитает сумму ячеек с A1 по *A10*. Чтобы передать ту же *формулу* в ячейку программно, нужно использовать ее англоязычное написание ([листинг 16.1](https://www.intuit.ru/studies/courses/494/350/lecture/8374?page=1#example.16.1).)

Range("B1").Formula = "=sum(A1:A10)"

Листинг 16.1. Запись формулы в ячейку

Как мы уже упоминали выше, есть особый *объект* - Application.WorksheetFunction - его методы представляют собой функции рабочего листа (более 250), которые можно использовать в коде *VBA*. Например, *функция* *Fact* вычисляет *факториал* переданного ей числа. Вот как выглядит работа с ней ([листинг 16.2](https://www.intuit.ru/studies/courses/494/350/lecture/8374?page=1#example.16.2).)

Dim num\_F As Integer

num\_F = InputBox("Введите число")

MsgBox ("Факториал числа " & num\_F & " равен " & \_

WorksheetFunction.Fact(num\_F))

Листинг 16.2. Использование формул MS Excel в коде VBA

В качестве аргументов функций можно использовать и объекты Range - то есть *диапазоны ячеек* или ячейки. Таким образом, например, можно проводить все необходимые расчеты в *VBA*, а на *лист* выгружать лишь готовые значения, без выгрузки формул.

**16.2. Работа с MS Excel из MS Word и наоборот**

[16-02-Excel to Word.xlsm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/16/files/16_02.xlsm) и [16-03-Word to Excel.docm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/16/files/16_03.docm) - примеры к п. 16.2.

Давайте рассмотрим взаимодействие различных приложений друг с другом. Сначала, мы займемся переносом данных из книги Microsoft *Excel* в документ Microsoft *Word*, затем - переносом данных в обратном направлении и использованием ресурсов *Excel* для проведения расчетов.

Напомню, что для такого взаимодействия нам понадобится подключить соответствующую объектную модель в редакторе *VBA* с помощью средства Tools o References.

Напишем *макрос* в *MS Excel*, который копирует содержимое выделенного диапазона в Microsoft *Word*, причем каждая строка диапазона собирается в одну текстовую строку, части которой, представляющие собой содержимое отдельных ячеек, разделяются запятыми и пробелами, а отдельные текстовые строки разделяются знаками перевода строки. Перед данными из таблицы выводится строка, содержащая информацию об имени книги и имени листа, откуда взята *информация* ([листинг 16.3](https://www.intuit.ru/studies/courses/494/350/lecture/8374?page=1#example.16.3).)

'Переменная для хранения ссылки

'на экземпляр MS Word

Dim obj\_Word As Word.Application

'Для хранения ссылки на документ

'MS Word

Dim obj\_WDoc As Word.Document

'Для хранения ссылки на лист

'книги Excel

Dim obj\_ESheet As Worksheet

'Для хранения ссылки на

'выделеный диапазон

Dim obj\_Range As Range

'Для хранения ссылки на

'книгу MS Excel

Dim obj\_Excel As Workbook

'Для строки формирования вывода

Dim str\_Str As String

'Записываем в переменные

'ссылки на соответствующие им

'объекты

Set obj\_Range = Selection

Set obj\_ESheet = ActiveSheet

Set obj\_Excel = ActiveWorkbook

'Запускаем экземпляр MS Word

Set obj\_Word = New Word.Application

'Делаем его видимым

obj\_Word.Visible = True

'Создаем новый документ

Set obj\_WDoc = obj\_Word.Documents.Add

'Активируем документ

obj\_WDoc.Activate

'Собираем строку для вывода информации

'об имени книги и листа

str\_Str = "Скопировано из книги " + \_

obj\_Excel.Name + ", с листа " + \_

obj\_ESheet.Name

'Выводим строку в документ

obj\_Word.Selection.TypeText (str\_Str)

obj\_Word.Selection.TypeParagraph

'Очищаем строку

str\_Str = ""

'Цикл для построчного прохода

'выделенного диапазона и сбора

'строк для вывода

For i = 1 To obj\_Range.Rows.Count

For j = 1 To obj\_Range.Columns.Count

str\_Str = str\_Str & obj\_Range.Cells(i, j) & ", "

Next j

'Удаляем из строки последних 2 символа

str\_Str = Mid(str\_Str, 1, Len(str\_Str) - 2)

'Активируем документ Word

obj\_WDoc.Activate

'Выводим текст и очищаем переменную

obj\_Word.Selection.TypeText (str\_Str)

obj\_Word.Selection.TypeParagraph

str\_Str = ""

Next i

Листинг 16.3. Работа с MS Word из MS Excel

Теперь напишем *макрос* в MS *Word*, выполняющий заявленные выше функции. Мы перенесем на *лист* *MS Excel* выделенный текст, разбив его на слова, после чего вычислим сумму нескольких чисел, используя ресурсы *MS Excel*. При этом слова будут скопированы каждое в отдельную ячейку, а весь этот материал будет размещен в таблице шириной 10 ячеек и высотой, которая зависит от количества слов в выделенном тексте ([листинг 16.4](https://www.intuit.ru/studies/courses/494/350/lecture/8374?page=1#example.16.4).).

'Объектные переменные для MS Excel

Dim obj\_Excel As Excel.Application

'Для книги

Dim obj\_Workbook As Excel.Workbook

'Для листа

Dim obj\_Worksheet As Excel.Worksheet

'Для текста в MS Word

Dim obj\_Range As Word.Range

'Для цикла, выгружающего слова на

'лист

Dim num\_Counter

'Для подсчета номера очередного слова

Dim num\_Words

'Присваиваем переменной ссылку

'на выделенную область

Set obj\_Range = Selection.Range

'Запустим MS Excel

Set obj\_Excel = New Excel.Application

'Добавим в Excel новую книгу

Set obj\_Workbook = obj\_Excel.Workbooks.Add

'Присвоим переменной ссылку на первый лист книги

Set obj\_Worksheet = obj\_Workbook.Worksheets(1)

'Вычислим результат от деления количества слов

'нацело - то есть - сколько полных строк

'по 10 слов удастся выделить из нашего текста

num\_Counter = obj\_Range.Words.Count \ 10

'Цикл выгрузки

For i = 1 To num\_Counter

For j = 1 To 10

'Увеличиваем на 1 номер

'выводимого слова

num\_Words = num\_Words + 1

'выводим это слово на лист

obj\_Worksheet.Cells(i, j) = \_

obj\_Range.Words.Item(num\_Words)

Next j

Next i

'Помимо полных строк по 10 слов

'может оказаться так, что некоторые

'слова образуют неполную строку

'вычислим ее длину

num\_Counter = obj\_Range.Words.Count - \_

(obj\_Range.Words.Count \ 10) \* 10

'Цикл выгрузки последних слов

For j = 1 To num\_Counter

num\_Words = num\_Words + 1

obj\_Worksheet.Cells(i, j) = \_

obj\_Range.Words.Item(num\_Words)

Next j

'Настроим ширину столбцов, содержащих

'выгруженные слова

obj\_Worksheet.Columns("A:J").AutoFit

'Теперь - вычисления

MsgBox ("Сумма чисел 1,2,3,4 равна " & \_

obj\_Excel.WorksheetFunction.Sum(1, 2, 3, 4))

MsgBox ("Данные выгружены в книгу " + \_

obj\_Workbook.Name)

'В конце работы отобразим

'MS Excel, до этого скрытый

obj\_Excel.Visible = True

Листинг 16.4. Работа с MS Excel из MS Word

**16.3. Работа с базами данных**

В примерах этого раздела используется *файл* Database.accdb, который должен быть расположен в корневом каталоге диска C.

Для работы с базами данных могут быть использованы различные инструменты. Одним из распространенных инструментов такого взаимодействия являются QueryTable - таблицы, которые отображают информацию, полученную из *базы данных*.

**16.3.1. OpenDatabase и QueryTable**

[16-04-Excel OpenDatabase.xlsm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/16/files/16_04.xlsm) - пример к п. 16.3.1.

Самый простой и доступный способ импортировать информацию из базы данных в Microsoft Excel, это - воспользоваться специальным методом рабочего листа. Речь идет о методе OpenDatabase. Он предназначен для создания новой книги, которая содержит лист с информацией, полученной из базы данных. Получение информации из базы данных в Excel может быть полезным, например, для анализа этой информации средствами Excel.

Полный вызов метода выглядит так:

OpenDatabase(Filename, CommandText, CommandType, BackgroundQuery, ImportDataAs)

Рассмотрим параметры метода.

* Filename - имя и расположение базы данных.
* CommandText - Текст запроса к базе данных. Здесь можно указать *имя таблицы* базы данных, которая должна быть открыта.
* CommandType - тип запроса - xlCmdCube (куб), xlCmdList (список), xlCmdSql (SQL), xlCmdTable (таблица).
* BackgroundQuery - если установлен в True - обработка данных ведется в фоновом режиме, если в False - в обычном.
* ImportDataAs - способ импорта данных. Может принимать два значения - первое - xlPivotTableReport (данные будут импортированы в виде сводной таблицы - *Pivot Table* ), второе - xlQueryTable (данные будут импортированы с помощью QueryTable - в виде обычной таблицы).

Чтобы рассмотреть пример использования этой команды, создадим простую базу данных, состоящую из двух таблиц. Первая таблица представляет собой список клиентов, вторая - список их покупок, где учитывается лишь сумма покупки на определенную дату. Таблица клиентов имеет имя Клиенты, таблица покупок - имя Покупки. Импортируем с помощью метода OpenDatabase таблицу Покупки в документ *MS Excel*. Предположим, что база данных хранится на диске C:, ее имя - Database.accdb. Добавим на лист *MS Excel* кнопку, содержащую такой код ([листинг 16.5](https://www.intuit.ru/studies/courses/494/350/lecture/8374?page=2#example.16.5).)

Workbooks.OpenDatabase \_

Filename:="C:\Database.accdb", \_

CommandText:="Покупки", \_

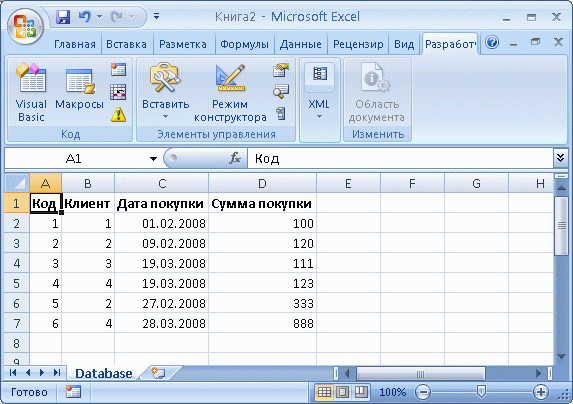
CommandType:=xlCmdTable, \_

BackgroundQuery:=True, \_

ImportDataAs:=xlQueryTable

Листинг 16.5. Импорт данных из БД с использованием QueryTable

После нажатия на кнопку будет создана новая книга, лист которой, названный по имени базы данных, будет содержать импортированные данные ([рис. 16.1](https://www.intuit.ru/studies/courses/494/350/lecture/8374?page=2#image.16.1).).



**Рис. 16.1.**QueryTable в документе

Чтобы импортировать данные как PivotTable, нам понадобится такой код ([листинг 16.6](https://www.intuit.ru/studies/courses/494/350/lecture/8374?page=2#example.16.6).) - его мы добавим в обработчик события Click другой кнопки на рабочем листе книги-примера.

Workbooks.OpenDatabase \_

Filename:="C:\Database.accdb", \_

CommandText:="Покупки", \_

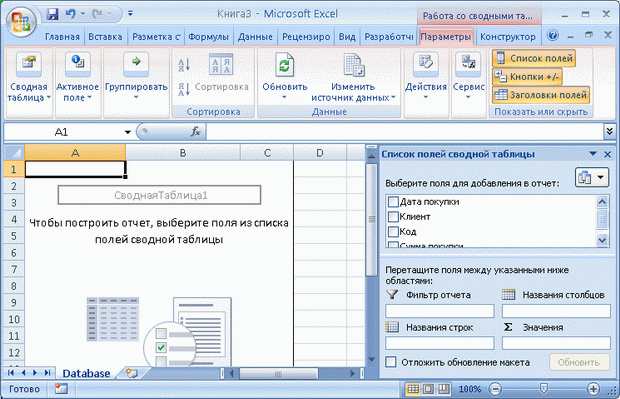
CommandType:=xlCmdTable, \_

BackgroundQuery:=True, \_

ImportDataAs:=xlPivotTableReport

Листинг 16.6. Импорт данных из БД с использованием PivotTable

На [рис. 16.2](https://www.intuit.ru/studies/courses/494/350/lecture/8374?page=2#image.16.2). вы можете видеть результат выполнения команды - сводную таблицу, с которой можно продолжать дальнейшую работу.

[](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/16/files/16_02.gif)

[увеличить изображение](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/16/files/16_02.gif)  
**Рис. 16.2.**Сводная таблица в документе MS Excel

Теперь рассмотрим еще один метод получения информации из БД.

**16.3.2. ADO**

[16-05-Excel ADODB Query.xlsm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/16/files/16_05.xlsm) - пример к п. 16.3.2.

QueryTable можно добавить на рабочий лист, предварительно настроив ее параметры.

Объекты QueryTable объединены в коллекцию QueryTables. Важнейший метод этой коллекции - Add - он добавляет новую таблицу в указанную позицию на листе. Вызов метода Add выглядит так:

*WorkBook*.QueryTables.Add(Connection, Destination)

В качестве параметра Connection обычно используют объект ADODB.Recordset, о котором ниже, а Destination - это объект Range, который указывает на диапазон (или ячейку), куда будет добавлена QueryTable. Если в Destination задана ячейка, левая верхняя ячейка вставляемой таблицы таблицы совпадет с ячейкой.

Для работы с базами данных используется объектная модель ADO. Чтобы подключить ее к проекту, выберите в окне References пункт Microsoft *ActiveX Data Object* 2.8 Library - обращаться к ней можно, используя имя объекта ADODB.

ADO - это очень мощный механизм для доступа к источникам данных. Здесь мы рассмотрим методику получения информации из БД с использованием ADO. Нас будут интересовать несколько ключевых объектов ADO.

Во-первых - это объект ADODB.Connection, который позволяет установить соединение с базой данных и работать с ней. У объекта Connection есть свойство ConnectionString - оно представляет собой строку, содержащие параметры *подключения к базе данных*, в частности - адрес файла базы данных и имя драйвера. Метод Open объекта Connection используется для открытия соединения, заданного свойством ConnectionString.

Во-вторых - объект ADODB.RecordSet - он позволяет получать из открытой базы данных определенные порции информации.

Для получения данных используется метод объекта Open, которому передается запрос на получение данных, а так же - открытое соединение.

Давайте рассмотрим пример. Здесь мы подключаемся к базе данных и создаем Query Table на основе объекта RecordSet, в котором хранится информация, полученная из базы ([листинг 16.7](https://www.intuit.ru/studies/courses/494/350/lecture/8374?page=2#example.16.7).).

'Для хранения ссылки на

'Query Table

Dim obj\_Query As QueryTable

'Для ссылки на соединение с

'базой данных

Dim obj\_ADOConn As ADODB.Connection

'Для ссылки на набор записей,

'полученный из БД

Dim obj\_ADORec As ADODB.Recordset

'Если в ячейке A5 есть данные 'значит мы уже вставляли сюда Query Table

'если данных нет - начинаем работу с БД

If ActiveSheet.Range("A5") <> "" Then

MsgBox "Уже есть Query Table в этом диапазоне"

Else

'Создаем новое соединение

Set obj\_ADOConn = New ADODB.Connection

'Вносим в ConnectionString параметры

'соединения. В Provider - имя драйвера,

'который используется для доступа к

'БД, а так же - параметр Data Source

'который отвечает за адрес источника данных

obj\_ADOConn.ConnectionString = \_

"Provider=Microsoft.ACE.OLEDB.12.0;" & \_

"Data Source=C:\Database.accdb"

'Подключаемся к базе данных

obj\_ADOConn.Open

'Создаем новый объект RecordSet 'Он хранит результат запроса к БД

Set obj\_ADORec = New ADODB.Recordset

'Выполняем запрос

'В параметре Source хранится строка

'SQL-запроса

'В ActiveConnnection - открытое соединение

obj\_ADORec.Open \_

Source:="SELECT \* FROM Покупки", \_

ActiveConnection:=obj\_ADOConn

'Создаем новую Query Table, в качестве

'источника данных передаем заполненный

'данными RecordSe

Set obj\_Query = \_

ActiveSheet.QueryTables.Add \_

(obj\_ADORec, Range("A5"))

'Используем метод Refresh для того

'чтобы таблица, заполненная данными

'была отображена на листе

obj\_Query.Refresh

End If

Листинг 16.7. Импорт данных из БД с использованием ADO

Если вы хотите эффективно работать с базами данных - вам придется научиться строить SQL-запросы, изучить особенности взаимодействия с различным видами БД и так далее.

**16.4. Работа с диаграммами**

[16-06-Excel Chart.xlsm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/16/files/16_06.xlsm) - пример к п. 16.4.

Для работы с *диаграммами* используют *объект* *Chart*. Чтобы добавить *диаграмму* на *лист* можно применить методом AddChart коллекции Shapes..

Такой код ([листинг 16.8](https://www.intuit.ru/studies/courses/494/350/lecture/8374?page=2#example.16.8).) добавляет *диаграмму* на *активный* *лист*:

ActiveSheet.Shapes.AddChart

Листинг 16.8. Добавление диаграммы на активный лист

Когда *диаграмма* добавлена, можно настроить ее свойства, в частности, при помощи метода SetSourceData задать *диапазон* (*объект* типа Range ), содержащий информацию, которая должна быть визуализирована. Этот метод принимает два параметра. Первый - Source - отвечает за *источник данных*, второй - PlotBy - определяет, как берутся данные для *диаграммы* - по столбцам ( xlColumns ) или по строкам ( xlRows ).

Так же после добавления *диаграммы* обычно настраивают ее тип - это делается с помощью свойства CharType. Оно может принимать одно из более чем 70 значений типа xlChartType. Например, xlConeCol - это трехмерная коническая *диаграмма*, xlPie - круговая *диаграмма*, xlLineMarkers - *график* с маркерами.

Рассмотрим пример ([листинг 16.9](https://www.intuit.ru/studies/courses/494/350/lecture/8374?page=2#example.16.9).). Добавим на *рабочий лист* обычную линейную *диаграмму*, используя *диапазон* значений, выделенных пользователем.

'Для хранения ссылки

'на диаграмму

Dim obj\_Chart As Chart

'Для хранения ссылки на

'диапазон входных значений

Dim obj\_Range As Range

Set obj\_Range = Selection

'Добавляем новую диаграмму и

'тут же выделяем ее

ActiveSheet.Shapes.AddChart.Select

Set obj\_Chart = ActiveChart

'Настраиваем исходные данные

'для диаграммы

obj\_Chart.SetSourceData \_

Source:=obj\_Range, \_

PlotBy:=xlRows

'Устанавливаем тип для диаграммы

obj\_Chart.ChartType = xlLine

Листинг 16.9. Добавление диаграммы на лист Excel

**16.5. Выводы**

В этой лекции мы рассмотрели некоторые дополнительные возможности программирования для *MS Excel*. Наше следующее занятие посвящено практическим примерам программирования для *MS Excel*.

Лекция 17. Практика MS Excel

**17.1. Система учета домашних финансов**

[17-01-Система учета домашних финансов.xlsm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/17/files/17_01.xlsm) - пример к п. 17.1.

*MS Excel* - это отличная среда для создания программ, автоматизирующих разного рода расчеты, для *математического моделирования* и т.д.. Давайте рассмотрим пример реализации простой системы учета домашних финансов.

**17.1.1. Условие**

Создадим в Excel простую систему учета домашних финансов. Она должна выполнять следующие функции:

* Предоставлять пользователю интерфейс для ввода и просмотра данных.
* Позволять вести учет доходов и расходов с возможностью указать источник дохода или расхода, сумму, и автоматическим указанием даты внесенной записи.
* Позволять исправлять ошибки в сумме записи или в информации по записи
* Уметь выводить текущий баланс доходов и расходов

Сразу же хочется отметить, что подобная система может быть расширена огромным количеством функций. Здесь мы приводим лишь основные блоки. При необходимости вы можете самостоятельно модифицировать их, приведя систему в нужное вам состояние. Например, вашу систему вполне можно оснастить средством для построения отчетов в MS Word - для этого вы можете воспользоваться методами работы, которые мы рассматривали выше.

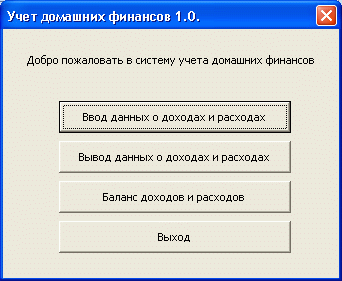
**17.1.2. Решение: создаем формы**

Создадим в проекте Microsoft Excel следующие формы ([табл. 17.1](https://www.intuit.ru/studies/courses/494/350/lecture/8375?page=1#table.17.1).)

|  |  |
| --- | --- |
| Таблица 17.1. Формы в проекте | |
| **Имя формы** | **Назначение** |
| frm\_Main | Организация доступа к другим формам программы |
| frm\_In | Ввод информации о доходах и расходах |
| frm\_Out | Построчный вывод информации о доходах и расходах |
| frm\_Balance | Вывод баланса доходов и расходов на текущую дату |

В [табл. 17.2](https://www.intuit.ru/studies/courses/494/350/lecture/8375?page=1#table.17.2) вы можете найти информацию об элементах управления на форме frm\_Main. На [рис. 17.1](https://www.intuit.ru/studies/courses/494/350/lecture/8375?page=1#image.17.1). приведен внешний вид формы.

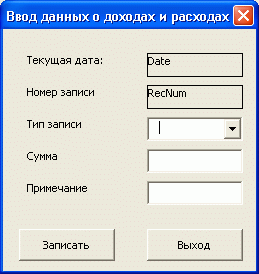
|  |  |
| --- | --- |
| Таблица 17.2. Элементы управления на форме frm\_Main | |
| **Имя и тип элемента управления** | **Назначение** |
| lbl\_Info | Информация о программе |
| cmd\_frm\_In | Вызов формы frm\_In |
| cmd\_frm\_Out | Вызов формы frm\_Out |
| cmd\_frm\_Info | Вызов формы frm\_Info |
| cmd\_Exit | Выход из программы |



**Рис. 17.1.**Форма frm\_Main

В [табл. 17.3](https://www.intuit.ru/studies/courses/494/350/lecture/8375?page=1#table.17.3). вы можете видеть информацию об элементах управления формы frm\_In ([рис. 17.2](https://www.intuit.ru/studies/courses/494/350/lecture/8375?page=1#image.17.2).)

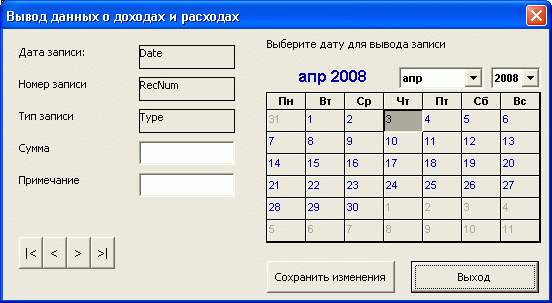
|  |  |
| --- | --- |
| Таблица 17.3. Элементы управления на форме frm\_In | |
| **Имя и тип элемента управления** | **Назначение** |
| lbl\_Date | Информация о текущей дате |
| lbl\_RecNum | Информация о номере записи |
| cbo\_Type | *Тип записи* - доход или расход |
| txt\_Sum | Сумма, в рублях |
| txt\_Info | Примечание |
| cmd\_Rec | Запись новой строки в файл |
| cmd\_Exit | Выход из формы |



**Рис. 17.2.**Форма frm\_In

В [табл. 17.4](https://www.intuit.ru/studies/courses/494/350/lecture/8375?page=1#table.17.4). вы можете видеть информацию об элементах управления формы frm\_Out ([рис. 17.3](https://www.intuit.ru/studies/courses/494/350/lecture/8375?page=1#image.17.3).)

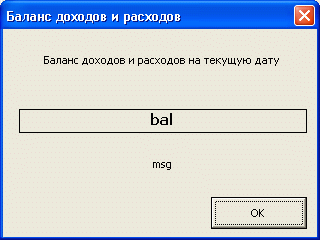
|  |  |
| --- | --- |
| Таблица 17.4. Элементы управления на форме frm\_Out | |
| **Имя и тип элемента управления** | **Назначение** |
| lbl\_Date | Информация о дате записи |
| lbl\_RecNum | Информация о номере записи |
| lbl\_Type | *Тип записи* - доход или расход |
| txt\_Summ | Сумма, в рублях |
| txt\_Info | Примечание |
| cmd\_Rec | Запись исправленных данных по текущей записи в файл |
| cmd\_Exit | Выход из формы |
| сmd\_First | Перейти на первую запись в таблице |
| сmd\_Last | Перейти на последнюю запись в таблице |
| сmd\_Forward | Перейти на следующую запись |
| сmd\_Backward | Перейти на предыдущую запись |
| сld\_First | Установить дату для вывода первой записи на эту дату |



**Рис. 17.3.**Форма frm\_Out

В [табл. 17.5](https://www.intuit.ru/studies/courses/494/350/lecture/8375?page=1#table.17.5). вы можете найти информацию об элементах управления формы frm\_Balance ([рис. 17.4](https://www.intuit.ru/studies/courses/494/350/lecture/8375?page=1#image.17.4).)

|  |  |
| --- | --- |
| Таблица 17.5. Элементы управления на форме frm\_Balance | |
| **Имя и тип элемента управления** | **Назначение** |
| lbl\_Balance | Баланс доходов и расходов на текущую дату |
| lbl\_Msg | Сообщение системы после анализа баланса |
| cmd\_OK | Кнопка OK |

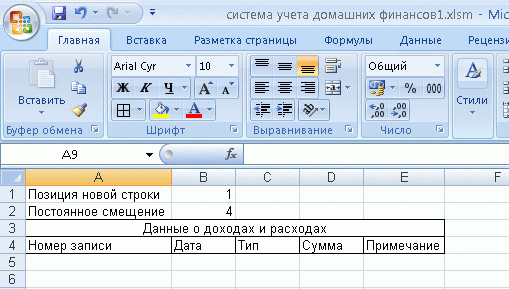


**Рис. 17.4.**Форма frm\_Balance

После того, как созданы формы, подготовим книгу Microsoft Excel для записи материалов.

**17.1.3. Подготовка книги Microsoft Excel**

В данном примере мы будем использовать один из листов книги в качестве хранилища данных. Давайте удалим из книги все листы кроме одного, назовем этот лист sh\_Base и для удобства создадим на листе заголовок таблицы, которая будет использоваться при работе программы. Вот, что у нас получилось ([рис. 17.5](https://www.intuit.ru/studies/courses/494/350/lecture/8375?page=1#image.17.5).).



**Рис. 17.5.**Структура данных для хранения информации

В строках таблицы "Данные о доходах и расходах" будут храниться записи, введенные пользователем с помощью формы frm\_In.

Для работы с этой таблицей мы будем использовать стиль ссылок R1C1, то есть, обращаться к ней по номеру строки и столбца. Ориентироваться внутри строк нам поможет знание следующих фактов о нашей таблице:

* Ширина таблицы составляет 5 ячеек.
* Номер записи - ячейка №1
* Дата - ячейка №2
* Тип - ячейка №3
* Сумма - ячейка №4
* Примечание - ячейка №5

Например, для того, чтобы узнать тип операции, записанной в строку с номером n нам понадобится проанализировать третью ячейку строки.

Для того, чтобы перемещаться по отдельным строкам таблицы, нам нужно знать, адреса первой и последней строк в таблице. Обратите внимание на то, что данные, которые будет вводить пользователь, будут располагаться начиная со строки №5, четыре первых строки заняты служебной информацией. То есть, первая строка таблицы будет располагаться в пятой строке листа Excel. Адресовать эту строку можно по-разному. Мы выбрали следующий способ: строка будет адресоваться собственным номером и постоянным смещением.

В ячейке листа B2 будем хранить информацию о постоянном смещении нашей таблицы. Там записано 4. Для того, чтобы получить номер строки листа, в котором хранится строка нашей таблицы с номером n, нужно n прибавить к значению постоянного смещения. То есть, для первой строки мы получим 4+1=5, для второй - 4+2=6.

В ячейке B1 хранится очень важное значение - мы назвали его Позиция новой строки. Это - адрес строки, которая должна быть записана при следующем обращении к процедуре записи. Для того, чтобы узнать номер последней заполненной строки, нужно из показателя Позиция новой строки вычесть 1.

Давайте подытожим сказанное. Для того, чтобы получить доступ к строке таблицы, используем номер этой строки и постоянное смещение. Адресовать отдельные ячейки в строке мы можем, зная структуру таблицы.

Ниже эти сведения помогут нам создать процедуры программы для работы с данными, а сейчас давайте рассмотрим "оснащение" каждой из форм.

**17.1.4. Код формы frm\_Main**

Для удобства здесь и далее код, относящийся к одной форме, приводится в таком виде, в котором он хранится в модуле формы - с названиями обработчиков событий и т.д. Выше мы подробно документировали состав каждой формы, это позволит вам легко ориентироваться в листингах. В [листинге 17.1](https://www.intuit.ru/studies/courses/494/350/lecture/8375?page=1#example.17.1). вы можете найти код формы frm\_Main.

Private Sub cmd\_Exit\_Click()

When\_Exit

End Sub

Private Sub cmd\_frm\_Balance\_Click()

frm\_Balance.Show

End Sub

Private Sub cmd\_frm\_In\_Click()

frm\_In.Show

End Sub

Private Sub cmd\_frm\_Out\_Click()

frm\_Out.Show

End Sub

Private Sub cmd\_frm\_Report\_Click()

frm\_Report.Show

End Sub

Private Sub UserForm\_Terminate()

When\_Exit

End Sub

Sub When\_Exit()

ThisWorkbook.Save

ThisWorkbook.Close

End Sub

Листинг 17.1. Код формы frm\_Main

Обратите внимание на то, что при нажатии кнопки cmd\_Exit, а так же - по событию UserForm\_Terminate(), которое происходит при закрытии главной формы, вызывается процедура When\_Exit. Она сохраняет *рабочую книгу* и закрывает ее. Таким образом, выйдя из главной формы, пользователь закрывает и книгу с данными.

При открытии книги мы отображаем на экране главную форму программы. Для этого мы добавили обработчик события Open для объекта *Workbook* ([листинг 17.2](https://www.intuit.ru/studies/courses/494/350/lecture/8375?page=1#example.17.2).). Напомню, что в браузере проектов объект *Workbook* называется ЭтаКнига.

Private Sub Workbook\_Open()

frm\_Main.Show

End Sub

Листинг 17.2. Код обработчика события Open для рабочей книги

Таким образом, открывая книгу, мы отображаем форму и не даем пользователю доступ к листу, закрывая форму, мы закрываем и книгу, что, опять же, не дает пользователю возможности вручную редактировать данные. Эти ограничения можно обойти. Например, в ходе разработки этой программы вам понадобится править ее код, анализировать таблицу с данными. Поэтому, если вы нажмете сочетание клавиш Ctrl+Pause Break - выполнение программы остановится, вы сможете редактировать код, вручную работать с таблицей.

Если вы будете создавать подобный проект для практических целей - возможно, для того, чтобы максимально усложнить доступ пользователя к данным, есть смысл зашифровать их. К сожалению, парольная защита проектов в MS Office встроенными средствами не слишком эффективна, поэтому вы либо должны быть готовы к потенциальному несанкционированному доступу, либо должны предусмотреть дополнительные меры безопасности. Хотя, надо отметить, что практически любую систему безопасности можно либо обойти, либо взломать.

Теперь давайте рассмотрим код элементов управления формы frm\_In.

**17.1.5. Код формы frm\_In**

[Листинг 17.3](https://www.intuit.ru/studies/courses/494/350/lecture/8375?page=2#example.17.3) содержит код формы frm\_In.

Private Sub cmd\_Exit\_Click()

frm\_In.Hide

'Скрывая frm\_In мы автоматически

'переходим к frm\_Main

End Sub

Private Sub cmd\_Rec\_Click()

'Адрес строки для записи

Dim num\_Address

'Вычисляем номер строки для записи

num\_Address = ActiveSheet.Range("B1") + \_

ActiveSheet.Range("B2")

'Записываем номер в первую ячейку строки

ActiveSheet.Cells(num\_Address, 1) = \_

ActiveSheet.Range("B1")

'Запишем дату во вторую ячейку

ActiveSheet.Cells(num\_Address, 2) = \_

Date

'В третьей ячейке - тип операции

ActiveSheet.Cells(num\_Address, 3) = \_

cbo\_Type.Value

'В четвертой - сумма

ActiveSheet.Cells(num\_Address, 4) = \_

Val(txt\_Sum)

'В пятой - примечание

ActiveSheet.Cells(num\_Address, 5) = \_

txt\_Info

'Запишем новый номер строки

ActiveSheet.Range("B1") = \_

ActiveSheet.Range("B1") + 1

'Сбросим все установки на форме

Initial

End Sub

Private Sub UserForm\_Activate()

'При активации формы

'инициализируем элементы управления

Initial

End Sub

Sub Initial()

'Инициализация элементов управления

lbl\_Date = Date

lbl\_RecNum = ActiveSheet.Range("B1")

cbo\_Type.Clear

cbo\_Type.AddItem "Доход"

cbo\_Type.AddItem "Расход"

cbo\_Type.Value = "Доход"

txt\_Info = ""

txt\_Sum = ""

End Sub

Листинг 17.3. Код формы frm\_In

Рассмотрим код формы frm\_Out

**17.1.6. Код формы frm\_Out**

[Листинг 17.4](https://www.intuit.ru/studies/courses/494/350/lecture/8375?page=2#example.17.4) содержит код формы frm\_Out. Обратите внимание на *пользовательскую процедуру* Load\_Data(). Мы передаем ей параметр num\_Index - номер строки, который должен быть отображен. Работа обработчиков нажатия на кнопки перемещения и обработчика, выполняющегося при выборе даты на календаре сводится к вычислению нужного номера строки и вызову этой процедуры.

Private Sub UserForm\_Initialize()

'Загружаем первую строку

Load\_Data (1)

End Sub

Private Sub cmd\_Backward\_Click()

'Предыдущая строка

If Val(lbl\_RecNum) > 1 Then

Load\_Data (Val(lbl\_RecNum) - 1)

End If

End Sub

Private Sub cmd\_Exit\_Click()

frm\_Out.Hide

End Sub

Private Sub cmd\_First\_Click()

'Загружаем первую строку

Load\_Data (1)

End Sub

Private Sub cmd\_Forward\_Click()

'Следующая строка

If Val(lbl\_RecNum) < ActiveSheet.Range("B1") Then

Load\_Data (Val(lbl\_RecNum) + 1)

End If

End Sub

Private Sub cmd\_Last\_Click()

'Загружаем последнюю строку

Load\_Data (ActiveSheet.Range("B1") - 1)

End Sub

Private Sub cld\_First\_Click()

'Просматриваем таблицу,

'находим первую запись

'за выбранную дату и выводим эту запись

For i = 1 To ActiveSheet.Range("B1") - 1

If ActiveSheet.Cells \_

(i + ActiveSheet.Range("B2"), 2) = \_

cld\_First.Value Then

Load\_Data (i)

Exit For

End If

Next i

End Sub

Private Sub cmd\_Rec\_Click()

'Адрес строки для записи

Dim num\_Address

'Вычисляем номер строки для записи

num\_Address = Val(lbl\_RecNum + \_

ActiveSheet.Range("B2"))

'Так как мы разрешили модифицировать

'лишь сумму и примечание - запишем их

'в текущую строку

'Запишем сумму

ActiveSheet.Cells(num\_Address, 4) = \_

Val(txt\_Sum)

'Запишем примечание

ActiveSheet.Cells(num\_Address, 5) = \_

txt\_Info

End Sub

Sub Load\_Data(num\_Index As Integer)

'Принимает номер строки и выводит

'Данные из этой строки

'Адрес строки для чтения

Dim num\_Address

'Вычисляем номер строки для чтения

num\_Address = num\_Index + \_

ActiveSheet.Range("B2")

'Выводим номер записи

lbl\_RecNum = \_

ActiveSheet.Cells(num\_Address, 1)

'Выводим дату

lbl\_Date = \_

ActiveSheet.Cells(num\_Address, 2)

'Выводим тип операции

lbl\_Type = \_

ActiveSheet.Cells(num\_Address, 3)

'Выводим сумму

txt\_Sum = \_

ActiveSheet.Cells(num\_Address, 4)

'Выводим примечание

txt\_Info = \_

ActiveSheet.Cells(num\_Address, 5)

End Sub

Листинг 17.4. Код формы frm\_Out

Теперь рассмотрим код модуля формы frm\_Balance

**17.1.7. Код формы frm\_Balance**

В [листинге 17.5](https://www.intuit.ru/studies/courses/494/350/lecture/8375?page=2#example.17.5). вы можете найти код модуля формы frm\_Balance. Здесь мы вычисляем баланс доходов и расходов по всей таблице. Вычисления ведутся в коде обработчика события Activate.

Private Sub cmd\_OK\_Click()

frm\_Balance.Hide

End Sub

Private Sub UserForm\_Activate()

'Адрес строки

Dim num\_Address

'Переменная для хранения суммы доходов

Dim num\_Earn

'Переменная для хранения суммы расходов

Dim num\_Spend

For i = 1 To ActiveSheet.Range("B1") - 1

num\_Address = i + ActiveSheet.Range("B2")

'Если в строке хранится значение дохода

'добавим его в num\_Earn

If ActiveSheet.Cells(num\_Address, 3) = "Доход" \_

Then

num\_Earn = num\_Earn + \_

ActiveSheet.Cells(num\_Address, 4)

End If

'Если в строке хранится значение расхода

'добавим его в num\_Spend

If ActiveSheet.Cells(num\_Address, 3) = "Расход" \_

Then

num\_Spend = num\_Spend + \_

ActiveSheet.Cells(num\_Address, 4)

End If

Next i

lbl\_Balance = num\_Earn - num\_Spend

If num\_Earn > num\_Spend Then \_

lbl\_Msg = "Доходы больше расходов."

If num\_Earn = num\_Spend Then \_

lbl\_Msg = "Доходы равны расходам."

If num\_Earn < num\_Spend Then \_

lbl\_Msg = "Доходы меньше расходов."

End Sub

Листинг 17.5. Код формы frm\_Balance

**17.2. Задача об обмене значениями**

[17-02-Обмен значений.xlsm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/17/files/17_02.xlsm) - пример к п. 17.2.

**17.2.1. Условие**

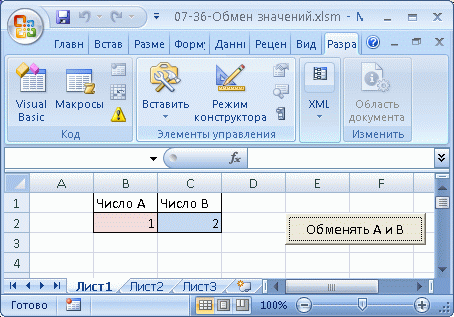
Произвести обмен значениями двух переменных без использования третьей

**17.2.2. Решение**

Предположим, что имеются 2 переменные (А и В), содержащие числа. Для обмена значениями этих переменных достаточно произвести следующие действия:

1. Сложить А и В и результат записать в А
2. Вычесть из А переменную В и записать результат в В.
3. Вычесть из А переменную В и записать результат в А.

Для решения задачи будем считать, что число A записано в ячейку B2, число В - в ячейку C2. Подпишем соответствующим образом эти ячейки и разместим на рабочем листе кнопку с именем cmd\_Change и надписью Обменять А и В ([рис. 17.6](https://www.intuit.ru/studies/courses/494/350/lecture/8375?page=3#image.17.6).)



**Рис. 17.6.**Рабочий лист, подготовленный для решения задачи

В [листинге 17.6](https://www.intuit.ru/studies/courses/494/350/lecture/8375?page=3#example.17.6). вы можете найти программный код для решения задачи, размещенный в обработчике события Click для кнопки cmd\_Change

'Сохраняем сумму ячеек в B2

ActiveSheet.Range("B2") = \_

ActiveSheet.Range("B2") + \_

ActiveSheet.Range("C2")

'Разность сохраняем в С2

ActiveSheet.Range("C2") = \_

ActiveSheet.Range("B2") - \_

ActiveSheet.Range("C2")

'И еще раз разность в B2

ActiveSheet.Range("B2") = \_

ActiveSheet.Range("B2") - \_

ActiveSheet.Range("C2")

Листинг 17.6. Решение задачи

**17.3. Перевод чисел из одной системы счисления в другую**

[17-03-Системы счисления.xlsm](https://www.intuit.ru/EDI/03_01_18_12/1514931930-14492/tutorial/517/objects/17/files/17_03.xlsm) - пример к п. 17.3.

**17.3.1. Условие**

Перевести заданное пользователем целое число A из одной системы счисления ( P ) в другую ( Q ). P и Q могут изменяться от 2 до 10.

**17.3.2. Решение**

*MS Excel* содержит функции для преобразования чисел между различными системами счисления, однако здесь мы создадим универсальное решение. Вы сможете встроить этот код в свои проекты. При необходимости вы сможете доработать его так, чтобы он смог работать с системами счисления, основание которых превышает 10.

В данном случае наиболее очевидным является перевод введенного числа сначала из системы счисления с основанием P в систему с основанием 10, а потом уже из системы с основанием 10 в систему с основанием Q.

Перевод в десятичную систему счисления осуществляется в два этапа:

1. Разбиение введенного числа на отдельные цифры
2. Получение десятичного числа по следующему алгоритму:

Первую цифру числа в системе счисления с основанием Q нужно умножить на Q в нулевой степени. Полученное число прибавить к произведению второй цифры числа, умноженную на Q в 1-й степени и так далее. Например:

Перевод из двоичной системы в десятичную:

1101 =1\*2^3+1\*2^2+0\*2^1+1\*2^0=8+4+0+1=13

Перевод из пятеричной системы в десятичную:

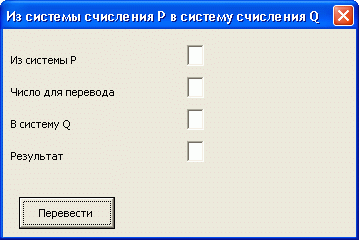
1042=1\*5^3+0\*5^2+4\*5^1+2\*5^0=125+0+20+2=147

Перевод в систему счисления с основанием Q из системы с основанием 10 осуществляется путем накапливания остатков от деления этого числа на Q с последующим изменением этого числа целочисленным делением его на Q до тех пор, пока переводимое число не станет равным 0.

Для решения этой задачи нам понадобится форма, содержащая следующие элементы управления ([табл. 17.6](https://www.intuit.ru/studies/courses/494/350/lecture/8375?page=3#table.17.6).). У текстовых полей свойство *AutoSize* установлено в True.

|  |  |
| --- | --- |
| Таблица 17.6. Элементы управления | |
| **Имя элемента управления** | **Подпись и примечания** |
| cmd\_OK | **Перевести**. Кнопка для перевода чисел |
| txt\_P | **Из системы P**. Текстовое поле для хранения основания системы счисления P. |
| txt\_Q | **В систему Q**. Поле для хранения основания системы счисления Q |
| txt\_A | **Число для перевода**. Число, заданное для перевода из системы P в Q |
| txt\_B | **Результат**. Текстовое поле для вывода результата перевода |

На [рис. 17.7](https://www.intuit.ru/studies/courses/494/350/lecture/8375?page=3#image.17.7). вы можете увидеть форму программы.



**Рис. 17.7.**Форма программы для перевода чисел из одной системы счисления в другую

В [листинге 17.7](https://www.intuit.ru/studies/courses/494/350/lecture/8375?page=3#example.17.7). вы можете найти код события Click для кнопки cmd\_OK.

'Для хранения основания системы P

Dim num\_P

'Для хранения основания системы Q

Dim num\_Q

'Для хранения числа в 10-й системе

Dim num\_10

'Для хранения очередного числа в

'системе счисления Q

Dim num\_S

num\_P = Val(txt\_P)

num\_Q = Val(txt\_Q)

'Переводим введенное число из

'системы счисления P

'в десятичную систему

For i = 1 To Len(txt\_A)

num\_10 = num\_10 + Val(Mid(txt\_A, i, 1)) \* \_

num\_P ^ (Len(txt\_A) - i)

Next i

'Переводим число из десятичной системы

'в систему с основанием Q

txt\_B = ""

While num\_10 <> 0

'Остаток от деления запишем в str\_S

num\_S = num\_10 Mod num\_Q

'Запишем очередное число в

'окно для вывода результата

txt\_B = Mid(Str(num\_S), 2, 1) + txt\_B

'Запишем в num\_10 результат

'целочисленного деления

'num\_10 на основание системы

'счисления Q

num\_10 = num\_10 \ num\_Q

Wend

Листинг 17.7. Решение задачи

**17.4. Выводы**

В этой лекции мы рассмотрели несколько практических примеров решения задач для *MS Excel*.

Дополнительный материал 1. Задания

Ниже приведены задачи, с помощью которых вы можете оценить ваш уровень владения материалом. Задания первого уровня подразумевают *знание* наиболее общих вопросов программирования на *VBA*, основ работы с элементами управления. Если вы справились с задачами первого уровня – это значит, что вы находитесь на начальном этапе освоения *VBA*.

Задачи второго уровня сложнее, они подразумевают более серьезные знания.

Задачи третьего уровня можно решить, хорошо освоив курс.

**1 уровень сложности**

1. Создать в редакторе VBA форму, добавить на неё кнопку и текстовое поле с подписью "Введите ваше имя". После запуска формы пользователь должен ввести своё имя в поле. По нажатию кнопки должно вывестись окно сообщения с текстом: "Здравствуйте, ИМЯ\_ПОЛЬЗОВАТЕЛЯ", где вместо "ИМЯ\_ПОЛЬЗОВАТЕЛЯ" должно быть подставлено введенное имя.
2. Создать в редакторе VBA форму, подписать её "Калькулятор". Добавить на форму три текстовых поля с подписями: "Число 1", "Число 2", "Сумма" и кнопку. По нажатию на кнопку в поле "Сумма" должна выводиться сумма чисел, введенных в поля "Число 1" и "Число 2".
3. Создать в редакторе VBA форму и разместить на ней текстовое поле, метку и кнопку. По нажатии на кнопку, текст, введенный в текстовом поле, должен отображаться на метке.
4. Создать в редакторе VBA форму, добавить на неё кнопку и метку. По нажатии на кнопку программа должна запрашивать у пользователя его имя, после чего выводить в метку текст: "Здравствуйте, ИМЯ\_ПОЛЬЗОВАТЕЛЯ", где вместо "ИМЯ\_ПОЛЬЗОВАТЕЛЯ" должно быть подставлено введенное имя.
5. Создать в редакторе VBA форму высотой 100 пикселей и шириной 100 пикселей. Разместить на форме, в её верхней части, две кнопки. Кнопку №1 подписать как "Увеличить", кнопку №2 – как "Восстановить". Написать обработчики событий Click для кнопок таким образом, чтобы при нажатии на кнопку №1 размеры формы изменялись бы до 300х400 пикселей, при нажатии на вторую кнопку – восстанавливались до 100х100 пикселей. После изменения размеров формы должно выводиться сообщение о её текущих размерах.
6. Создать в редакторе VBA форму, разместить на ней кнопку и текстовое поле. По нажатию на кнопку в текстовое поле должно выводиться сообщение, содержащее информацию о ширине формы такого вида: "Ширина текущей формы – x пикселей", где вместо x должна быть подставлена ширина формы.

**2 уровень сложности**

1. Создать в редакторе VBA форму и разместить на ней два текстовых поля. Добавить к ним подписи "Введите число" и "Вывод". Запретите изменение пользователем содержимого второго поля. При вводе в первое окно числа во втором окне должно выводиться слово "Чётное" если число четное и "Нечетное" если число нечетное. Если введено не число – во втором окне должна быть отображена фраза "Неверный ввод".
2. Создать в редакторе VBA форму размером 300х400 пикселей. Добавить на форму кнопку размером 25х25 пикселей. Написать обработчик соответствующего события для этой кнопки, который, при наведении указателя мыши на кнопку перемещает её в случайную позицию в пределах формы, а так же выводит в заголовок формы информацию о количестве перемещений кнопки.
3. Создать в редакторе VBA форму, добавить на нее 5 элементов управления CheckBox, пронумеровать их от 1 до 5, задать имена вида cbo\_n, где n – номер элемента управления от 1 до 5. Добавить на форму четыре кнопки. Первая кнопка, подписанная как "Установить четные", устанавливает четные флажки, кнопка "Сбросить все" сбрасывает все флажки, кнопка "Количество установленных флажков" выводит сообщение вида "Количество установленных флажков равно n ", в том случае, если ни один флажок не установлен, выводит сообщение "Ни один флажок не установлен"
4. Добавить в VBA-проект три формы. Первую форму назвать frm\_Main, вторую – frm\_1, третью – frm\_2. Разместить на форме frm\_Main три кнопки. Две из них должны скрывать frm\_Main и отображать, соответственно, frm\_1 и frm\_2, третья должна завершать работу программы. На frm\_1 и frm\_2 разместить по одной кнопке, которая, соответственно, скрывает frm\_1 и frm\_2 и отображает главную форму. Разместить в документе MS Word кнопку, которая отображает главную форму.
5. Написать программу, которая запрашивает у пользователя число n, большее 1, после чего заполняет одномерный массив длиной n чисел случайными числами в диапазоне от 1 до n\*3 и выводит этот массив в документ MS Word, разделив отдельные элементы массива запятой
6. Написать программу, которая запрашивает у пользователя число n от 2 до 20, после чего заполняет массив длиной n чисел, запрашивая ввод каждого из чисел у пользователя. После ввода всех элементов , программа выводит в окне сообщения максимальный элемент массива.
7. Написать программу, которая вводит массив случайных чисел в диапазоне от 1 до 100, после чего подсчитывает количество четных и нечетных элементов массива и выводит эти данные в окне сообщения.
8. Написать программу, которая вводит массив из n случайных чисел ( n задаёт пользователь) в диапазоне от 1 до 100, после чего вычисляет их сумму и выводит в окне сообщения.
9. Написать программу, которая вводит массив из n случайных чисел ( n задаёт пользователь) в диапазоне от -100 до 100, после чего вычисляет сумму четных элементов массива, абсолютное значение которых превышает 25.

**3 уровень сложности**

1. Разработать программу, которая запрашивает у пользователя два числа – A и B и возвращает результат деления A на B. Предусмотреть перехват и обработку ошибок времени выполнения.
2. Разработать программу, которая запрашивает у пользователя два числа, одно – длиной 4 цифры, не содержащее цифр 2, 4, 7, второе – длиной 5 чисел, не содержащее одинаковых цифр. Если пользователь ввел неправильные числа – программа должна запросить их ввод снова, предусмотрев отказ от ввода.
3. Разработать программу, которая запрашивает у пользователя количество слов, которое он хотел бы обработать, после чего создаёт массив символьных переменных, запрашивая у пользователя каждое из слов. После заполнения массива программа должна вывести в документ MS Word список введенных слов, а так же – информацию о том, какая буква алфавита (прописные и строчные буквы учитываются совместно) встречается чаще всего.
4. Разработать программу, которая предлагает пользователю выбрать длину одномерного массива – от 10 до 100 с шагом в 5. Заполняет этот массив случайными числами в диапазоне от 10 до 115, после чего преобразует его в двумерный массив (матрицу), в каждой из строк которой содержится по 5 элементов исходного массива, количество строк зависит от длины исходного массива, выбранного на начальном этапе. Перенести полученную матрицу на лист MS Excel, выделить цветом ячейки матрицы, содержащие четные числа, большие 50.
5. Написать макрос MS Word, который работает с участком текста документа, выделенного пользователем. А именно – заносит каждое слово участка текста в массив, удалив знаки препинания, сортирует массив по возрастанию и выводит в новый документ MS Word в таком формате: Номер слова; слово; длина слова
6. Разработать на VBA программу для шифрования и расшифровки русскоязычного текста, выделенного в текстовом редакторе MS Word сдвиговым методом на произвольное целое число символов - n, которое и будет паролем для расшифровки текста. Строчные и прописные буквы шифруются отдельно. При шифровании букв, коды которых находятся на расстоянии n+1 от конца кодового пространства строчных или прописных букв, использовать для замены символы из начала кодового пространства. Например, кодовое пространство строчных русскоязычных букв находится в диапазоне 224 – 255 – если нужно зашифровать символ с кодом 250 при размере n, равном 7, то этот символ заменяется на второй символ, т.е. 7- (255-250)=2. Другие символы в тексте должны остаться незашифрованными.