

Visual Basic for Applications



Variables, procedures and functions

user procedures and functions

arithmetic operations, variables and constants

exception handling

Materials

<http://staff.uz.zgora.pl/ipajak>

<http://staff.uz.zgora.pl/gpajak>

Summary of the Lecture01

- ❑ Application is a set of objects, each contains certain properties and methods.
- ❑ The most important Excel objects: **Application**, **Workbook**, **Worksheet**, **Range**.
- ❑ Each cell, selection, column, row, etc. is a **Range** object.
- ❑ Reference to property and method:

`object_name.property_name`

`object_name.method_name`

- ❑ **Application** object active **Workbook** and active **Worksheet** are default objects, and they can be omitted in references to object properties.
- ❑ Application contains collection of workbooks (object property **Workbooks**), each Workbook contains collection of its worksheets (object property **Worksheets**).
- ❑ Reference to collection:

`collection_name(index)`

- ❑ Each property has a type determining the range of values that it can store.
- ❑ To set a value of any element (e.g. property) assignment statement „=” can be used:

`element = value`

Basic principles of programming in VBA

- ❑ VB is case insensitive (additionally, text editor automatically corrects source code).
- ❑ The names defined by user must start with a letter, it can not contain special chars (space , * ; . = etc.).
- ❑ A single instruction should be placed on one program line.
- ❑ In order to split instruction into several program lines continuation symbol „ _” (space followed by underscore) should be used.
- ❑ If several instructions are placed in one program line they should be separated by „:” (reduces the readability of the code, it should be used only in special cases).
- ❑ Empty lines are omitted, they can be used to divide the code into smaller blocks (improves program readability).
- ❑ Language keywords (name of instructions, elements of macro headers, etc.) are reserved and cannot be used as names of elements defined by user.
- ❑ Text that begins with an apostrophe is a comment, it does not affect the execution of the program.

Notation

- ❑ **Keywords** (e.g. name of statement) – roman font, bold,
- ❑ `Predefined elements` (e.g. name of standard functions) – roman font,
- ❑ `[optional elements]` – square brackets,
- ❑ `element1|element2` – alternative elements (exactly one has to be chosen),
- ❑ `<name>` (e.g. name of macro) – text inside angle brackets, italic,
- ❑ *description* – italic.

Example

`[Private|Public] Sub <name>`

- **Private**, **Public** and **Sub** are keywords (bold)
- The first word is **Private** or **Public** (vertical bar), but it is optional (square brackets)
- The word **Sub** is obligatory
- After **Sub** there is a name created by the user (angle brackets)

Procedures (subroutines) in VBA

```
[Private|Public] Sub <name>([arg1, arg2, ... , argN])  
    code of procedure  
End Sub
```

- ❑ **Sub** is a keyword, it specifies the header of procedure.
- ❑ **Private** stands for private procedure, accessible only in one module.
- ❑ **Public** stands for public procedure, accessible in the whole program.
- ❑ **Private** and **Public** are optional, default procedure is public.
- ❑ <name> is a name defined by user (see point 2., s.3).
- ❑ *arg1, arg2, ... argN* is an optional list of arguments, defined as follows:

```
[Optional] <argument_name> As type [=default_value]
```
- ❑ **Optional** means optional argument, which can be omitted when procedure is called (in such case it takes *default_value*).
- ❑ Procedure executes subsequent instructions until **End Sub**
- ❑ Execution of procedure can be broken before its end using **Exit Sub**

Calling a procedure

`<procedure_name> [arg1, arg2, ... , argN]`

- ❑ There are no parentheses in the procedure call (arguments should be written after a space).
- ❑ `<procedure_name>` describes standard or user defined procedure accessible in active excel document (workbook).
- ❑ `arg1, ... argN` is an argument list corresponding to argument list defined in header of procedure (see s.5).
- ❑ Optional argument can be omitted, in such a case free space separated by comma should be remained.
- ❑ Arguments can be specified in any order, in such a case the name of argument should be given according to syntax:

`<argument_name> := value`

Example – MsgBox

Header

```
MsgBox(prompt As String, _  
        Optional buttons As VbMsgBoxStyle = vbOKOnly, _  
        Optional title As String = "Microsoft Excel")
```

- prompt – message displayed in the window
- buttons – sets of buttons visible in the window (default vbOKOnly – only OK button) and kind of window (displayed icon). Accessible values: vbCritical, vbQuestion, vbExclamation, vbInformation
- title – title of window (default „Microsoft Excel“)

Examples

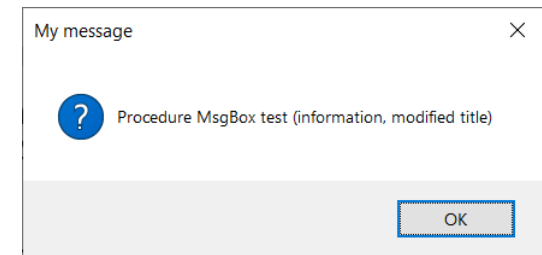
```
MsgBox "Procedure MsgBox test"
```

```
MsgBox "Procedure MsgBox test", vbCritical
```

```
MsgBox "Procedure MsgBox test", vbQuestion, "My message"
```

```
MsgBox "Procedure MsgBox test", , "My message"
```

```
MsgBox buttons:=vbExclamation, title:=" My message", _  
        prompt:="Procedure MsgBox test"
```



Assignment statement, arithmetic operators

Assignment statement

<element> = expression

Assignment statement allows to modify any element of program that has not been defined with the read-only attribute.

Note: the expression to the right side of the assignment statement is always executed first, and the result of the expression is assigned to the element on the left side.

Arithmetic operators

+	addition	\	integer division
-	subtraction	mod	modulo (remainder of a division)
*	multiplication	^	exponentiation
/	division	()	grouping of operations

Arithmetic operations are performed from the left to the right with priority (first exponentiation, next multiplication and division, then addition and subtraction). Operator symbol cannot be omitted, parentheses change the order of operations.

Note: $x + 5/3 * y = x + \frac{5}{3}y$, $(x + 5)/(3 * y) = \frac{x+5}{3y}$

Example

Task: The procedure gets the value from the currently selected cell, increments it by two, puts the result to the cell on the right (same row, next column) and selects it.

References

- Currently selected cell: `ActiveCell`
- Reference to the value of active cell: `ActiveCell.Value`
- Reference to the value of cell on the right: `ActiveCell.Offset(0,1).Value`
- Selection of cell on the right: `ActiveCell.Offset(0,1).Activate`

```
Public Sub IncrementToTheRight()
```

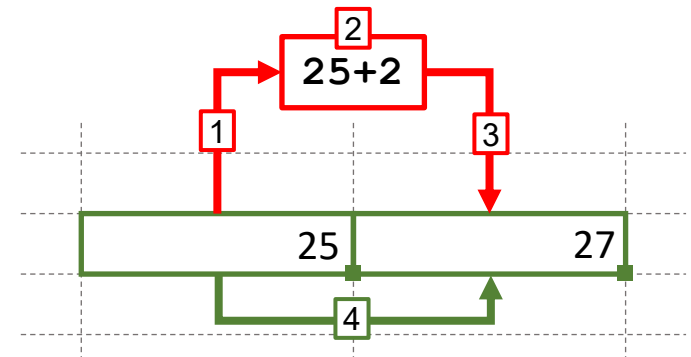
```
    With ActiveCell
```

```
        3. Offset(0, 1).Value = .Value + 2
```

```
        4. Offset(0, 1).Activate
```

```
    End With
```

```
End Sub
```



Source codes are available on the website

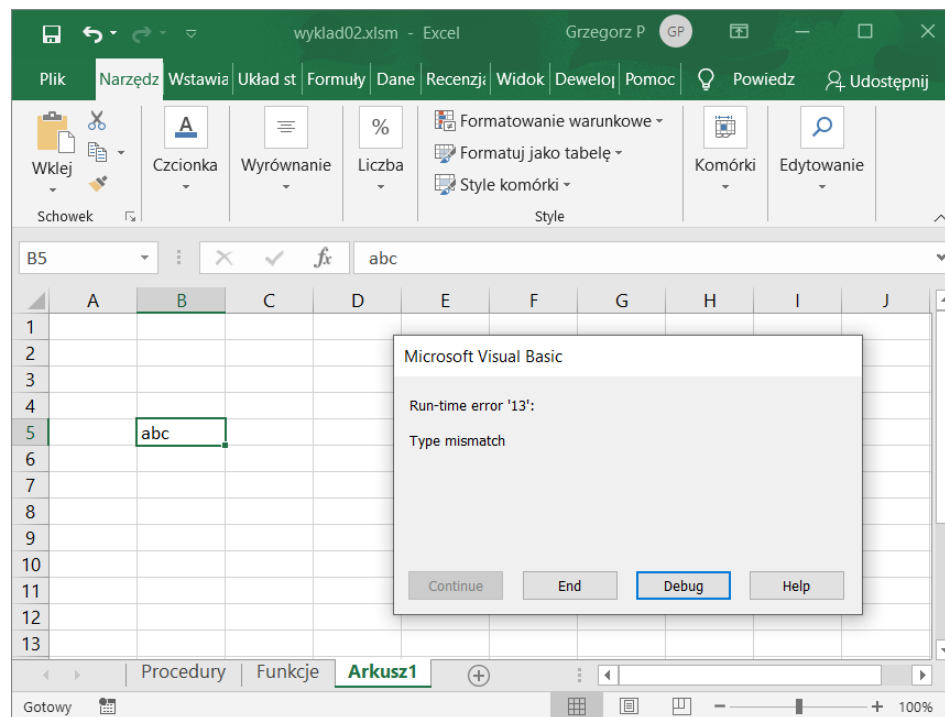
Run-time errors (exceptions)

Run-time error is an error that takes place while executing a program (in contrast to compilation errors that occur during compilation before running the program)

Example: arithmetic operations are defined only for numerical values, so the statement:

```
ActiveCell.Offset(0, 1).Value = ActiveCell.Value + 2
```

reports an error (raises exception), when active cell contains text (String value).



Exception handling

Exception handling

On Error GoTo <*label*>

If an error occurs the program stops current statement and goes to the location indicated by the *label*.

On Error Resume Next

If an error occurs the program skips current statement and executes the next one.

On Error GoTo 0

Cancels current **On Error** settings, and restores standard error handling.

<*label*> is an character sequence ended by ":" (colon), indicating the location in which program starts execution when an error occurs.

Resuming the program after error handling

Resume

The program resumes execution at the line where the error occurred.

Resume Next

The program resumes execution at the next line after the error occurred.

Example I

Supplementing the `IncrementToTheRight` procedure with exception handling – ver.1.
When error occurs procedure displays message box and stops.

```
Public Sub IncrementToTheRight1()
```

```
    On Error GoTo BadValue
```

```
    With ActiveCell
```

```
        .Offset(0, 1).Value = .Value + 2
```

```
        .Offset(0, 1).Select
```

```
    End With
```

```
    Exit Sub
```

```
BadValue:
```

```
    MsgBox "Select a numeric value", vbCritical, "Error"
```

```
End Sub
```

Note: The procedure executes subsequent statements to an `End Sub` instance. An `Exit Sub` statement, before the label, prevents displaying the message when an error has not occurred.

Source codes are available on the website

Example II

Supplementing the `IncrementToTheRight` procedure with exception handling – ver.2.
When error occurs procedure skips current statement and executes the next one.

```
Public Sub IncrementToTheRight2 ()
```

```
    On Error Resume Next
```

```
    With ActiveCell
```

```
        .Offset(0, 1).Value = .Value + 2
```

```
        .Offset(0, 1).Select
```

```
    End With
```

```
End Sub
```

Source codes are available on the website

In the case of cells containing text (String) the add operation causes an error, which will be ignored, so procedure executes next statement and selects next cell.

Worksheet row before first procedure execution

The same row after first procedure execution (value has not been changed, selection has been moved)

The same row after next procedure execution (value has been changed and selection has been moved)

	A	B	C
1	a	5	
1	a	5	
1	a	5	7

Example III version 1

Procedure copies a row in which cursor is placed to worksheet named "Copy". Successive copied rows are placed below the last one. Worksheet "Copy" should exist.

```
Public Sub CopyRow1()  
    On Error GoTo LostWorksheet  
    With Worksheets("Copy").UsedRange  
        ActiveCell.EntireRow.Copy .Rows(.Rows.Count + 1).EntireRow  
    End With  
    Worksheets("Copy").Columns("A:E").AutoFit  
    Exit Sub  
LostWorksheet:  
    MsgBox "The worksheet ""Copy"" is lost, vbCritical  
End Sub
```

No	Name	Amount	Invoice date	Payment date
1	A-Z Elementy dekoracyjne	2 300,00 zł	16.02.2022	22.03.2022
2	Hydrostal Sp. z o.o. ZP	980,20 zł	04.02.2022	16.06.2022
3	Kaiser Schody Sp. z o.o.	3 575,00 zł	15.02.2022	12.04.2022
4	RECTOR Polska Sp. z o.o.	227,50 zł	28.02.2022	19.06.2022
5	ComfortHouse sp. z o.o.	1 232,80 zł	05.02.2022	28.05.2022
6	AB KLIMA Chigo.pl	2 500,00 zł	06.02.2022	09.05.2022
7	AS Instrument Polska	128,75 zł	19.02.2022	06.04.2022
8	Bikrol-Stal s.c.	3 750,00 zł	18.02.2022	23.05.2022

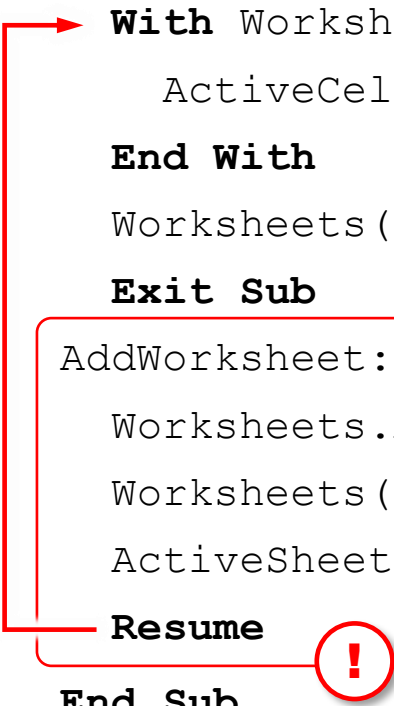
No	Name	Amount	Invoice date	Payment date
1	Kaiser Schody Sp. z o.o.	3 575,00 zł	15.02.2022	12.04.2022
2	AS Instrument Polska	128,75 zł	19.02.2022	06.04.2022

Source codes are available on the website

Example III version 2

Modification of the example from s.14. If worksheet "Copy" does not exist it is created.

```
Public Sub CopyRow2 ()  
    On Error GoTo AddWorksheet  
    With Worksheets ("Copy") .UsedRange  
        ActiveCell.EntireRow.Copy .Rows (.Rows.Count + 1) .EntireRow  
    End With  
    Worksheets ("Copy") .Columns ("A:E") .AutoFit  
    Exit Sub  
AddWorksheet:  
    Worksheets.Add (After:=Worksheets ("Invoices")) .Name = "Copy"  
    Worksheets ("Invoices") .Activate  
    ActiveSheet.Rows (1) .Copy Worksheets ("Copy") .Rows (1)  
    Resume  
End Sub
```



Note: Resume statement returns to the program line in which the error occurs.

Source codes are available on the website

Variables and constants

Variable – container for a data processed by program, identified by unique name. The value assigned to variable can be changed during program execution.

Constant – unique symbol representing some value (number, text, etc.). Value assigned to constant cannot be changed during program execution.

Variable declaration scheme

```
Dim <name> As type
```

Constant declaration scheme

```
Const <name> [As type] = value
```

<name> must be unique (there cannot be two items with the same name) and should fulfill the conditions described on s.3, point 2.

Examples

```
Dim x As Integer
```

```
Dim cell As Range
```

```
Const VAT As Single = 0.23
```

Note: Declaration of non-object variables is not required. Each undeclared variable is given type **Variant**. The declaration of variables can be forced using **Option Explicit** placed on the beginning of the module (**Tools -> Options -> Require variable declaration**).

Setting the value of variable

Setting the value of a non-object variable

`<name> = expression`

Setting the value of an object variable

Set `<name> = expression`

`<name>` is name of variable

`expression` is any VBA expression (including name of variable, constant, etc.) with a type compatible with the type of the variable.

Examples

```
Dim x As Integer
```

```
Dim y As Integer
```

```
Dim k As Range
```

```
x = 5
```

```
y = 29 + x ^ 2
```

```
Set k = Worksheets("Worksheet5").Range("C3:E5")
```

Example

The procedure swaps the value of two cells: the active cell and the cell to the right of it (the same row, next column).

Algorithm

1. Take the value of the active cell and put it to the variable.
2. Take the value of the cell on the right and put it to the active cell.
3. Put the value stored in the variable to the cell on the right.

```
Public Sub SwapToTheRight1()
```

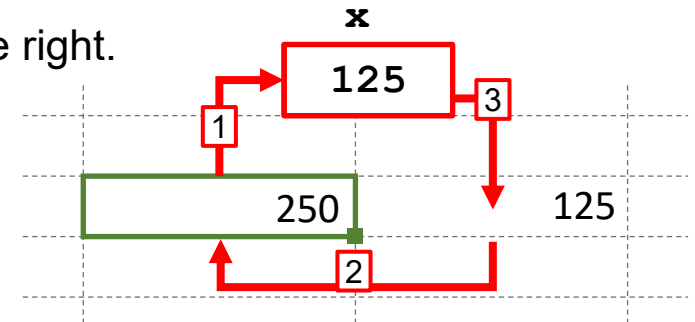
```
    Dim x As Integer
```

```
    x = ActiveCell.Value
```

```
    ActiveCell.Value = ActiveCell.Offset(0, 1).Value
```

```
    ActiveCell.Offset(0, 1).Value = x
```

```
End Sub
```



Source codes are available on the website

Note: Above procedure works correctly only for **Integer** values.

A problem to think about (1): how to swap values of any type.

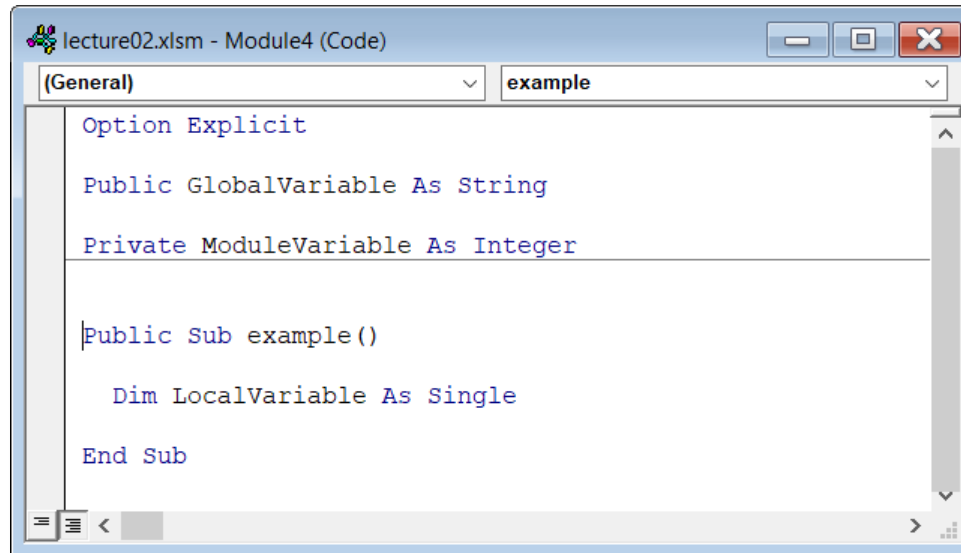
A problem to think about(2): how to swap given ranges.

Lifetime and scope of variables

Local variable – a variable declared inside VBA macro. It only exists when macro is executing and is removed when macro is completed.

Module variable – a variable declared in module using **Dim** or **Private** (recommended). It is available in all macros inside the module, exists for the entire duration of the program execution.

Global variable – a variable declared in module using **Public**. It is available in all modules, exists for the entire duration of the program execution.



```
lecture02.xlsm - Module4 (Code)
(General) example
Option Explicit

Public GlobalVariable As String

Private ModuleVariable As Integer

Public Sub example()

    Dim LocalVariable As Single

End Sub
```

Functions in VBA

```
[Private|Public] Function <name>([arg1,... , argN]) As type  
    code of function  
  
    <name> = value
```

End Function

- ❑ **Function** is a keyword, it specifies the header of function.
- ❑ **Private**|**Public** are optional, they determine the availability of the function, public function can be used in all modules and as worksheet function.
- ❑ <name> is a name defined by user (see point 2., s.3).
- ❑ *type* specifies the type of the value returned as the result of the function.
- ❑ Result of the function is specified by assigning *value* to <name>.
- ❑ *arg1,... argN* is an optional list of arguments, defined as in procedure (s.5).
- ❑ Function executes subsequent instructions until **End Function**
- ❑ Execution of function can be broken before its end using **Exit Function**
- ❑ Error handling is implemented in the same way as in procedure.

Calling of function

variable = *<function_name>*[(*arg1*, ... , *argN*)]

- ❑ The parentheses are required if arguments are passed. In the case of argument less function the parentheses are optional.
- ❑ *<function_name>* describes standard or user defined function accessible in active excel document (workbook).
- ❑ *variable* stands for the name of the variable where the value returned by the function will be stored.
- ❑ *arg1*, ... *argN* is an argument list corresponding to argument list defined in header of function (see s.20).
- ❑ Optional argument can be omitted, in such a case free space separated by comma should be remained.
- ❑ Arguments can be specified in any order, in such a case the name of argument should be given according to syntax:

<argument_name> := *value*

Communication with the user – InputBox

Function InputBox

```
InputBox(prompt As String,  
         Optional title As String = "Microsoft Excel",  
         Optional default As String = "", ...) As String
```

Method InputBox (class Application)

```
InputBox(prompt As String,  
         Optional title As Variant = "Microsoft Excel",  
         Optional default As Variant = "", ...  
         Optional type As Variant) As Variant
```

- `prompt` – text displayed inside the dialog window
- `title` – title of the dialog window (default „Microsoft Excel“)
- `default` – default value
- `type` – type of value to read (only method `InputBox`): 0 – formula, 1 – number, 2 – text (string), 4 – logical value, 8 – range of cell (object `Range`).

Note: Both function and method `InputBox` have four additional arguments, omitted in the above description (position of window and references to help).

Example – InputBox

Procedure fills selected range using the value specified by user. To select the range method InputBox and to determine the value function InputBox is used.

```
Public Sub FillRange()  
    Dim rng As Range  
    Dim val As String  
    On Error Goto Cancel  
    Set rng = Application.InputBox("Select range", type:=8)  
    val = InputBox("Enter value")  
    rng.Value = val  
    Exit Sub  
Cancel:  
End Sub
```

Note: Error handling ensures the correct reaction of the program in the case of pressing the Cancel button in the InputBox window. In this case, the method returns a empty value that cannot be assigned to the Range object variable.

Source codes are available on the website

User function – examples

```
Const DefaultVAT As Single = 0.23
```

```
Public Function GrossPrice1(net_price As Currency,  
                             vat As Single) As Currency
```

```
GrossPrice1 = net_price + net_price * vat
```

```
End Function
```

```
Public Function GrossPrice2(net_price As Currency,  
                             Optional vat As Single = DefaultVAT) As Currency
```

```
GrossPrice2 = net_price + net_price * vat
```

```
End Function
```

	A	B	C	D	E	F	G	H
1	Function GrossPrice1					Function GrossPrice2		
2	Net price	VAT	Gross price			Net price	Gross price	
3	127,00 zł	23%	156,21 zł			127,00 zł	156,21 zł	=GrossPrice2(F3)
4	250,00 zł	23%	307,50 zł			250,00 zł	307,50 zł	
5	35,00 zł	8%	37,80 zł			35,00 zł	37,80 zł	=GrossPrice2(F5;0,08)
6	875,00 zł	23%	1 076,25 zł			875,00 zł	1 076,25 zł	
7	1 230,00 zł	8%	1 328,40 zł			1 230,00 zł	1 328,40 zł	
8	27,00 zł	23%	33,21 zł			27,00 zł	33,21 zł	
9								

=GrossPrice1(A3;B3)

Source codes are available on the website