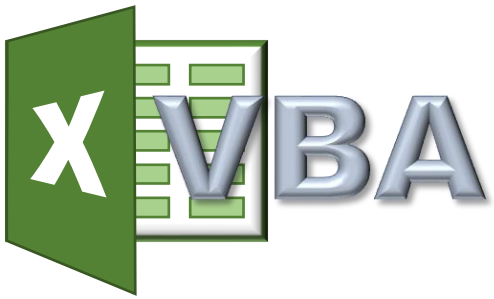
Techniki programowania



Obsługa błędów, typy makr, instrukcja warunkowa

obsługa błędów wykonania
procedury i funkcje użytkownika
projektowanie algorytmów
Instrukcja warunkowa

Podsumowanie Wykładu01

- Aplikacja jest zbiorem obiektów, każdy zawiera pewne właściwości i metody.
- Najważniejsze obiekty Excel-a: **Application**, **Workbook**, **Worksheet**, **Range.**
- Aplikacja może zawierać kolekcję Workbook-ów (obiekt **Workbooks**), jeden Workbook zazwyczaj zawiera kolekcję arkuszy (obiekt **Worksheets**).
- Każda komórka, zaznaczenie, kolumna, wiersz, itp. są obiektem typu **Range**.
- Odwołanie do właściwości obiektu (metoda analogicznie)

```
nazwa_obiektu.nazwa_właściwości
```

- Odwołanie do kolekcji

```
nazwa_kolekcji(indeks)
```

- Obiekty: **Application**, aktywny **Workbook** i aktywny **Worksheet** są domyślne, na ogół mogą być pominięte w odwołaniach.
- Do zmiany wartości dowolnego elementu (np. właściwości) służy „**=**”

```
element = wartość
```

- Operacje arytmetyczne: **+**, **-**, *****, **/**, ****, **^**, **mod**, **( )**
- Program przechowuje dane w zmiennych, schemat deklaracji

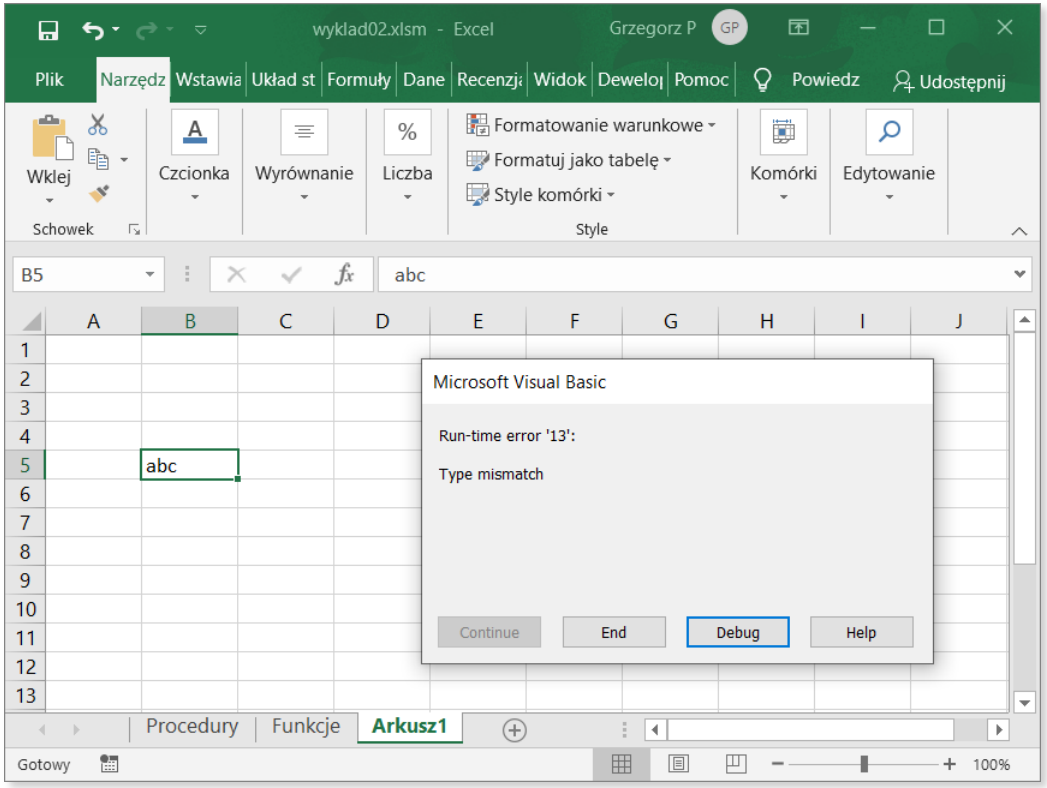
```
Dim <nazwa> As typ
```

Błędy wykonania

Operacje arytmetyczne są zdefiniowana tylko dla wartości liczbowych (liczba całkowita, rzeczywista, waluta lub data), więc instrukcja:

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

zgłasza błąd, jeżeli aktywna komórka zawiera tekst (wartość String).



Ustawienia obsługi błędów

`On Error GoTo` *`<etykieta>`*

W przypadku błędu następuje skok do wskazanego miejsca w programie

`On Error Resume Next`

W przypadku błędu program kontynuuje wykonie kodu od następnego wiersza

`On Error GoTo` `0`

Anuluje aktualne ustawienia **`On Error`**, przywraca standardową obsługę błędów VBA.

`<etykieta>` to ciąg znaków zakończony ”**:**” (dwukropek), który wskazuje wiersz od którego rozpocznie się wykonanie kodu po wystąpieniu błędu.

Wznawianie programu po obsłudze błędu

`Resume`

Procedura wznawia wykonanie od wiersza, w którym wystąpił błąd

`Resume Next`

Procedura wznawia wykonanie od następnego wiersza po tym, w którym wystąpił błąd

Przykład I

Uzupełnienie procedury `InkrementujWPrawo` (w.1, s.33) o obsługę błędów – wersja 1. W przypadku błędu procedura wyświetla okno komunikatu i kończy działanie.

```
Public Sub InkrementujWPrawo1()
  On Error GoTo NieprawidłowaWartość
  With ActiveCell
    .Offset(0, 1).Value = .Value + 2
    .Offset(0, 1).Select
  End With
  Exit Sub
NieprawidłowaWartość:
  MsgBox "Wybierz komórkę zawierającą liczbę", vbCritical, "Błąd"
End Sub
```

Uwaga: Procedura wykonuje instrukcje do wystąpienia **`End Sub`**. Instrukcja **`Exit Sub`** (przerwanie wykonywania procedury) umieszczona przed etykietą zapobiega wyświetlaniu komunikatu gdy błąd nie wystąpił.

Kod dostępny na stronie przedmiotu

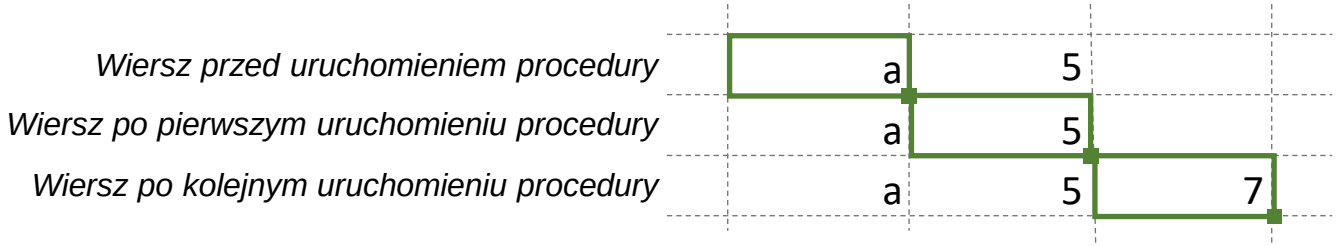
Przykład II

Uzupełnienie procedury `InkrementujWPrawo` (w.1, s.33) o obsługę błędów – wersja 2. W przypadku błędu procedura kontynuuje działanie od wiersza następnego.

```
Public Sub InkrementujWPrawo2()
  On Error Resume Next
  With ActiveCell
    .Offset(0, 1).Value = .Value + 2
    .Offset(0, 1).Select
  End With
End Sub
```

Kod dostępny na stronie przedmiotu

W przypadku komórek zawierających tekst operacja dodawania zgłasza błąd, który zostanie zignorowany (nie będzie wyświetlone okno komunikatu) i procedura wykona kolejną instrukcję (zaznaczenie komórki sąsiedniej).



Przykład III

Procedura kopiuje zawartość aktualnie wybranego zakresu do arkusza o nazwie "Kopia". Jeżeli arkusz "Kopia" nie istnieje zostanie utworzony.

```
Public Sub KopiujZakres()
  Dim a1 As Worksheet
  Dim a2 As Worksheet
  On Error GoTo DodajArkusz
  Worksheets("Kopia").Range(Selection.Address).Value =_
  Selection.Value
  Exit Sub
DodajArkusz:
  Set a1 = ActiveSheet
  Set a2 = Worksheets.Add()
  a2.Name = "Kopia"
  a1.Activate
  Resume
End Sub
```

Jeżeli arkusz "Kopia" nie istnieje wystąpi błąd

Obsługa błędu

Uwaga: po dodaniu nowego arkusza do kolekcji (metoda **Add**) Excel ustawia go jako aktywny, stąd konieczność "zapamiętania" aktualnie wybranego arkusza (zmienna **a1**) i powrót do niego przed zakończeniem bloku obsługi błędu.

Kod dostępny na stronie przedmiotu

Notacja

- **`Słowa kluczowe języka`** (np. nazwy instrukcji) – czcionka prosta, pogrubienie,
- `Elementy predefiniowane` (np. nazwy standardowych funkcji) – czcionka prosta, bez pogrubienia,
- `[elementy opcjonalne]` – nawiasy kwadratowe,
- Elementy alternatywne (należy wybrać jeden z nich) są rozdzielane pionową kreską (znak |),
- `<nazwa>` – tekst w nawiasach ostrych, kursywa,
- *`opis`* – kursywa.

Przykład

```
[Private|Public] Sub <nazwa>
```

- Pierwsze słowo to **`Private`** lub **`Public`**, jest opcjonalne (nawiasy kwadratowe)
- Słowo **`Sub`** musi wystąpić zawsze
- Po słowie **`Sub`** występuje nazwa wprowadzana przez użytkownika
- **`Private`**, **`Public`** i **`Sub`** to słowa kluczowe (pogrubienie)

```
[Private|Public] Sub <nazwa>([arg1, arg2, … , argN])
  miejsce na kod procedury
End Sub
```

- **Sub** jest słowem kluczowym języka określa nagłówek procedury (subroutine)
- **Private** oznacza procedurę prywatną, dostępną tylko w module
- **Public** oznacza procedurę publiczną, dostępną w dowolnym miejscu programu
- Słowa **Private** i **Public** są opcjonalne, domyślnie procedury są publiczne
- *<nazwa>* jest nazwą nadawaną przez użytkownika (patrz wykład 1., s.1-21, punkt 2.)
- *arg1, arg2, … argN* to opcjonalna lista argumentów zdefiniowanych jako:

  `[Optional] <nazwa_argumentu> As typ [= wartość_domyślna]`

- **Optional** oznacza argument opcjonalny, który może być pominięty podczas wywołania (w takim przypadku przyjmuje *wartość_domyślną*
- Procedura wykonuje kolejne instrukcje do wystąpienia **End Sub**
- Działanie procedury można przerwać umieszczając w kodzie instrukcję **Exit Sub**

Schemat wywołania procedury

```
<nazwa_procedury> [arg1, arg2, … , argN]
```

- W wywołaniu procedury nie występują nawiasy
- *<nazwa_procedury>* określa procedurę standardową lub zdefiniowaną przez użytkownika i dostępną w aktywnym arkuszu
- *arg1, … argN* odpowiadają kolejnym argumentom zgodnie z definicją zapisaną w nagłówku procedury
- Każdy argument opcjonalny może być pominięty (puste miejsce)
- Argumenty procedury można podać w dowolnej kolejności określając ich nazwy zgodnie ze schematem:

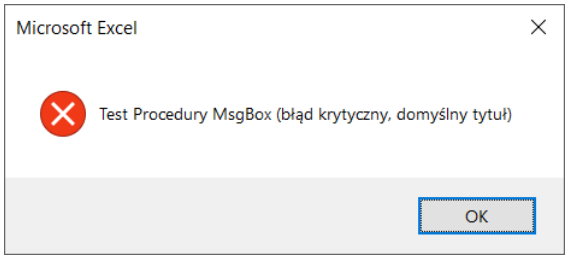
```
<nazwa_argumentu> := wartość
```

Przykład – MsgBox

Nagłówek

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

- `prompt` – tekst wyświetlany w oknie komunikatu
- `buttons` – zestaw przycisków dostępnych w oknie (`vbOKOnly` – tylko przycisk OK) oraz typ okna (wyświetlana ikona). Dostępne wartości: `vbCritical`, `vbQuestion`, `vbExclamation`, `vbInformation` (błąd krytyczny, pytanie, ostrzeżenie, informacja)
- `title` – tytuł okna (domyślnie „Microsoft Excel”)



Przykład wywołania

```
MsgBox "Test Procedury MsgBox"
MsgBox "Test Procedury MsgBox, vbCritical
MsgBox "Test Procedury MsgBox", vbQuestion, "Mój komunikat"
MsgBox "Test Procedury MsgBox", , "Mój komunikat"
MsgBox buttons:=vbExclamation, title:="Mój komunikat", _
       prompt:="Test Procedury MsgBox"
```

Kod dostępny na stronie przedmiotu

```
[Private|Public] Function <nazwa>([arg1,… , argN]) As typ
  miejsce na kod funkcji
  <nazwa> = wartość
End Function
```

- **Function** jest słowem kluczowym języka określa nagłówek funkcji
- **Private|Public** są opcjonalne, określają dostępność funkcji, funkcja publiczna jest traktowana jak standardowa funkcja arkuszowa i może być użyta w formułach
- *<nazwa>* jest nazwą nadawaną przez użytkownika (patrz wykład 1., s.1-21, punkt 2.)
- *typ* określa typ wartości zwracanej jako wynik działania funkcji
- Wynik funkcji jest określany przez przypisanie *wartości* do *<nazwy>*
- *arg1,… argN* to opcjonalna lista argumentów, definicja jak w procedurze (s.2-6)
- Funkcja wykonuje kolejne instrukcje do wystąpienia **End Function**
- Działanie funkcji można przerwać umieszczając w kodzie instrukcję **Exit Function**
- Obsługa błędów może być realizowana tak jak w przypadku procedur

Wywołanie funkcji

Schemat wywołania funkcji

```
zmienna = <nazwa_funkcji>[(arg1, … , argN)]
```

- W wywołaniu funkcji nawiasy są wymagane jeżeli następuje przekazanie argumentów. W przypadku funkcji bezargumentowej są dozwolone, ale niewymagane.
- `<nazwa_funkcji>` określa procedurę standardową lub zdefiniowaną przez użytkownika i dostępną w aktywnym arkuszu.
- `zmienna` oznacza nazwę zmiennej, w której zostanie umieszczona wartość zwracana przez funkcję.
- `arg1, … argN` odpowiadają kolejnym argumentom zgodnie z definicją zapisaną w nagłówku procedury.
- Każdy argument opcjonalny może być pominięty (puste miejsce)
- Argumenty funkcji można podać w dowolnej kolejności określając ich nazwy zgodnie ze schematem:

```
<nazwa_argumentu> := wartość
```

Komunikacja z użytkownikiem – InputBox

Funkcja `InputBox`

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

Metoda `InputBox` (klasa Application)

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

- `prompt` – tekst wyświetlany w oknie
- `title` – tytuł okna (domyślnie „Microsoft Excel”)
- `default` – wartość domyślna
- `type` – typ odczytywanej wartości (tylko metoda `InputBox`): 0 – formuła, 1 – liczba, 2 – tekst, 4 – wartość logiczna, 8 – zakres (**Range**).

Uwaga: Zarówno funkcja jak i metoda `InputBox` ma cztery dodatkowe parametry (położenie okna i odwołanie do systemu pomocy), które nie zostały opisane.

Przykład – InputBox

Procedura wypełnia wskazany zakres określoną wartością. Do odczytu zakresu wykorzystano metodę `InputBox`, do odczytu wartości funkcję `InputBox`.

```
Public Sub Wypełnij()
  Dim Zakres As Range
  Dim wartość As String
  On Error Goto Wycofanie
  Set Zakres = Application.InputBox("Wskaż zakres", type:=8)
  wartość = InputBox("Podaj wartość")
  Zakres.Value = wartość
  Exit Sub
  Wycofanie:
End Sub
```

błąd wykonania

Uwaga: Obsługa błędu zapewnia prawidłową reakcję programu w przypadku naciśnięcia przycisku Anuluj w oknie InputBox (metoda obiektu Application). W takiej sytuacji zwracana jest wartość pusta, która nie może być przypisana do zmiennej obiektowej Zakres (wartość pusta nie jest obiektem, więc próba przypisania prowadzi do zgłoszenia błędu wykonania), co powoduje skok do etykiety Wycofanie i zakończenie działania procedury.

Kod dostępny na stronie przedmiotu

Funkcje użytkownika – przykład

```
Const DomyślnyVAT As Single = 0.23

Public Function CenaBrutto1(netto As Currency,
                            vat As Single) As Currency
  CenaBrutto1 = netto + netto * vat
End Function

Public Function CenaBrutto2(netto As Currency,
                            Optional vat As Single = DomyślnyVAT) As Currency
  CenaBrutto2 = netto + netto * vat
End Function
```

	A	B	C	D	E	F	G	H
1	**Funkcja CenaBrutto1**					**Funkcja CenaBrutto2**		
2	**Netto**	**VAT**	**Brutto**			**Netto**	**Brutto**	
3	127,00 zł	23%	156,21 zł			127,00 zł	156,21 zł	
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	
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								

=CenaBrutto2(F3)

=CenaBrutto2(F5;0,08)

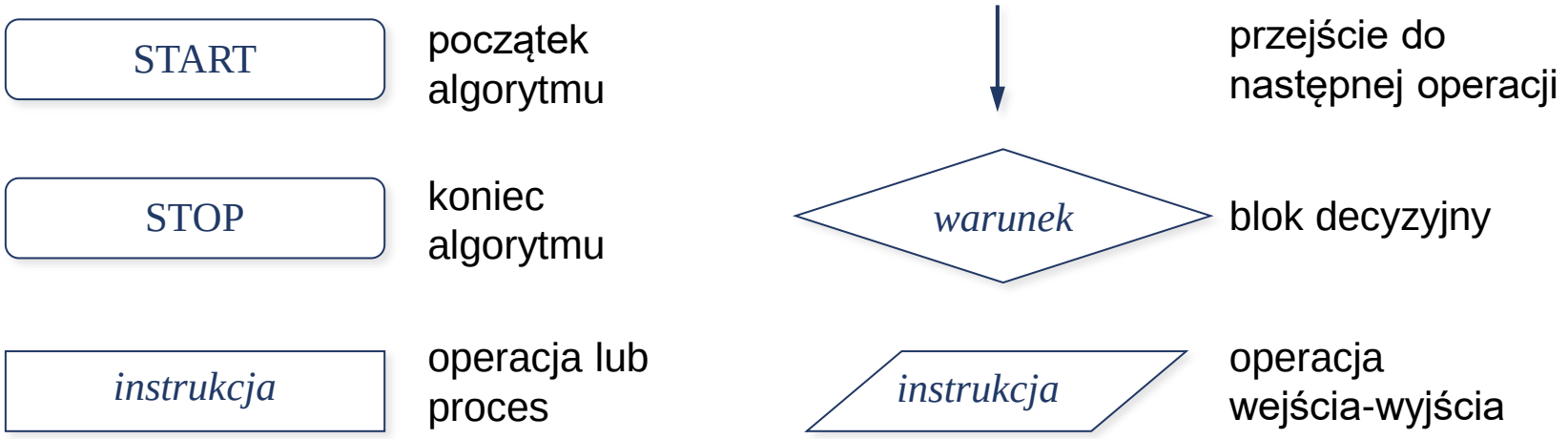
=CenaBrutto1(A3;B3)

Kod dostępny na stronie przedmiotu

Algorytm – przepis postępowania prowadzący do rozwiązania określonego zadania; zbiór poleceń określających sposób przetwarzania zbioru danych ze wskazaniem kolejności w jakiej mogą być wykonane.

Realizacja każdego programu powinna być poprzedzona procesem projektowania, którego istotnym elementem jest zaplanowanie sposobu przetwarzania danych. Algorytm stanowi uniwersalny język zapisu, niezależny od docelowego narzędzia programowania.

Podstawowe symbole



Przykład

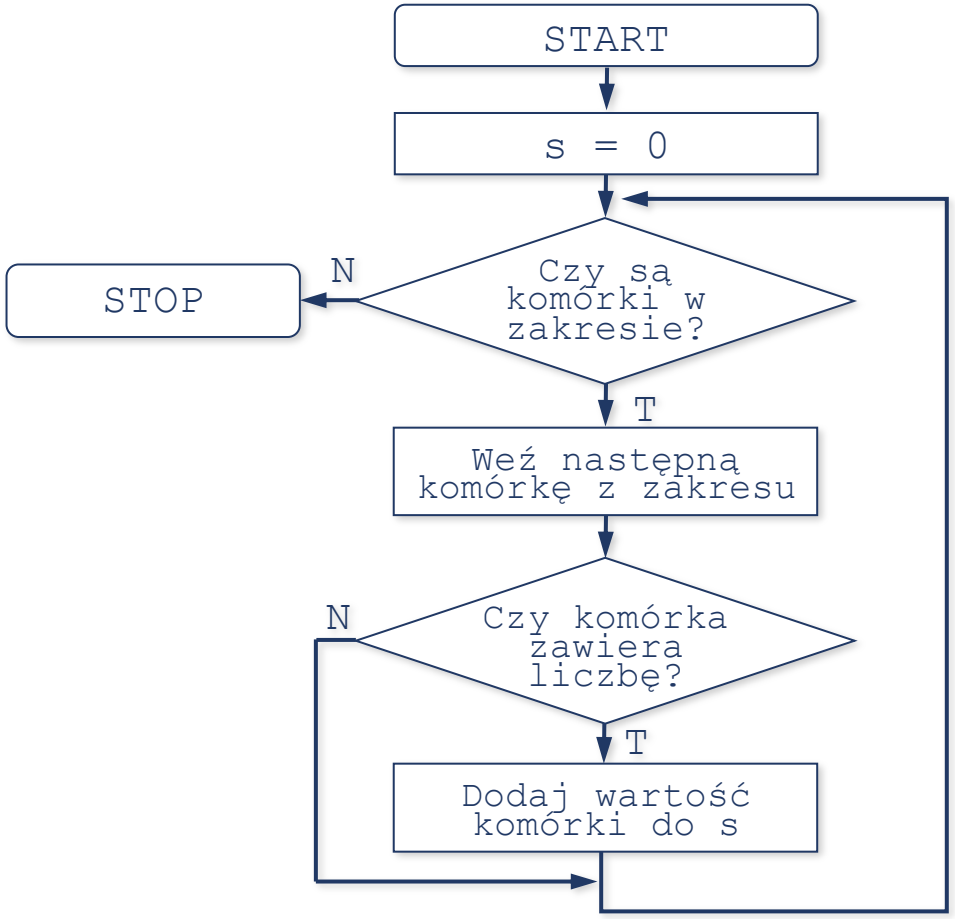
Zadanie: zaprojektować algorytm obliczający sumę wartości liczbowych w podanym zakresie arkusza.

	A	B	C
1	1		1
2	2		2
3	a		a
4	01.03.2022		01.03.2022
5	b		b
6	3		3
7	4		4
8	44631		10

=SUMA(A1:A7)

=SumaLiczb(C1:C7)

Uwaga: standardowa funkcja SUMA traktuje daty jak wartości liczbowe.



Instrukcje sterujące

Instrukcja sterująca – element języka programowania, który służy do określenia kolejności wykonania instrukcji zawartych w kodzie programu.

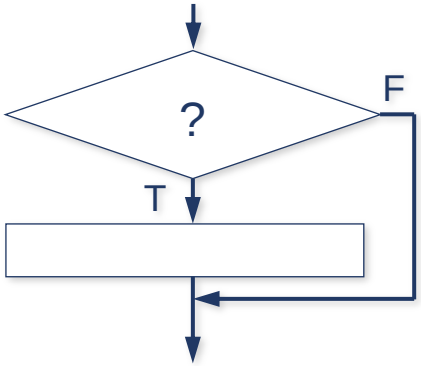
Instrukcje sterujące w VBA

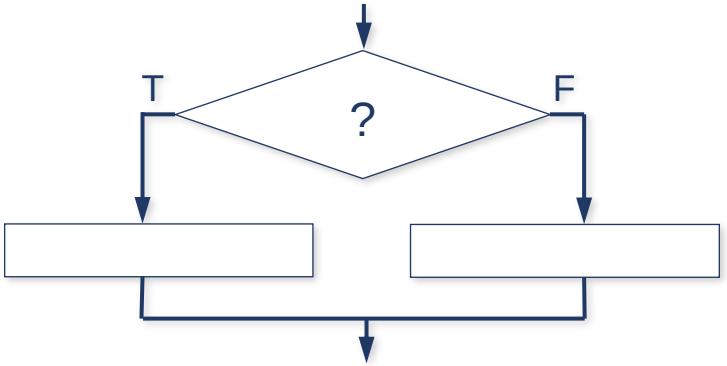
- **Instrukcja warunkowa** (**`If-Then`**, **`If-Then-Else`**) – wprowadza rozgałęzienie w kodzie programu, tworząc alternatywne sekwencje instrukcji.
- **Instrukcja wyboru** (**`Select Case`**) – wprowadza rozgałęzienie w kodzie programu tworząc dowolną liczbę alternatywnych sekwencji instrukcji.
- **Instrukcje iteracyjne (pętli)** (**`For-Each`**, **`For-Next`**, **`Do-Loop`**) – umożliwiają cykliczne powtarzanie sekwencji instrukcji.

Dodatkowo

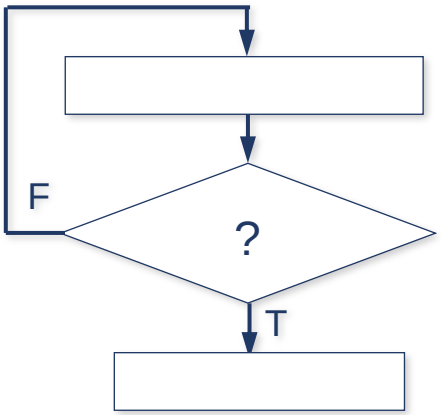
- **Funkcje wyboru wartości z listy** (**`Choose`**, **`Switch`**) – umożliwiają wybór wartości z predefiniowanej listy, zastępują szereg instrukcji warunkowych lub instrukcję wyboru.

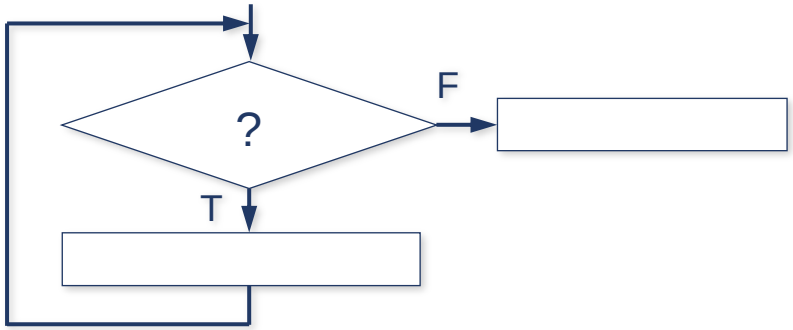
Instrukcje warunkowe





Instrukcje iteracyjne (pętli)

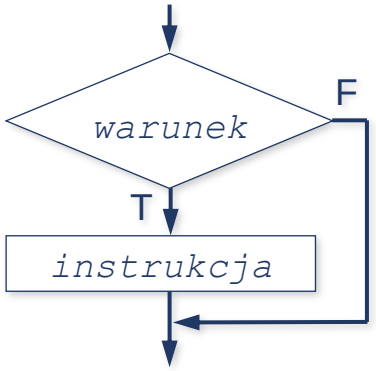




Instrukcja warunkowa

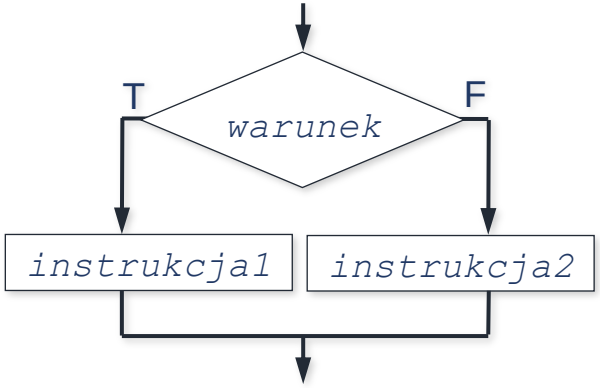
Jeżeli *warunek* jest prawdziwy wykonaj *instrukcję* (grupę instrukcji).

```
If warunek Then instrukcja

If warunek Then
  instrukcja1
  …
  instrukcjaN
End If
```



Jeżeli *warunek* jest prawdziwy wykonaj *instrukcję1* (pierwszą grupę instrukcji) w przeciwnym wypadku wykonaj *instrukcję2* (drugą grupę instrukcji).

```
If warunek Then instrukcja1 Else instrukcja2

If warunek Then
  instrukcja1-1
  …
  instrukcja1-N
Else
  instrukcja2-1
  …
  instrukcja2-N
End If
```



Operatory relacyjne i logiczne

Operatory relacyjne

- `>` większy
- `<` mniejszy
- `>=` większy lub równy
- `<=` mniejszy lub równy
- `<>` różny
- `=` równy

Operatory logiczne

- **`And`** koniunkcja
- **`Or`** alternatywa
- **`Not`** negacja

x	y	x And y	x Or y	Not x
False	False	False	False	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False

Przykład (x jest zmienną liczbową)

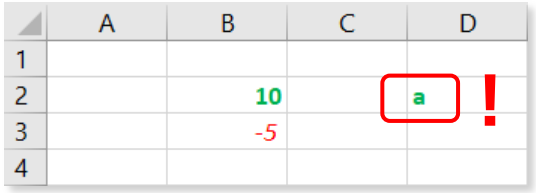
- Sprawdzenie czy x jest liczbą dodatnią: `x > 0`
- Sprawdzenie czy x jest różna od zera: `x <> 0`
- Sprawdzenie czy $x \in [-3,\ 5]$: `x >= -3 And x <= 5`
- Sprawdzenie czy $x \in (-\infty, -2] \cup (10,\ +\infty)$: `x <= -2 Or x > 10`

Procedura ustawia kolor zielony w komórkach zawierających wartości dodatnie i zero, czerwony w komórkach zawierających wartości ujemne.

```
Public Sub Koloruj1()
  If ActiveCell.Value >= 0 Then ActiveCell.Font.Color = vbGreen _
  Else ActiveCell.Font.Color = vbRed
End Sub
```

Procedura ustawia kolor zielony oraz pogrubienie w komórkach zawierających wartości dodatnie i zero, czerwony oraz kursywę w komórkach zawierających wartości ujemne.

```
Public Sub Koloruj2()
  If ActiveCell.Value >= 0 Then
    ActiveCell.Font.Color = vbGreen
    ActiveCell.Font.Bold = True
  Else
    ActiveCell.Font.Color = vbRed
    ActiveCell.Font.Italic = True
  End If
End Sub
```



Uwaga: tekst jest interpretowany jako liczba dodatnia.

Kod dostępny na stronie przedmiotu

Sprawdzenie typu zmiennej

Wybrane funkcje informacyjne VBA

- `IsDate(w)` – określa czy `w` może być konwertowane na typ `Date`,
- `IsEmpty(w)` – określa czy `w` jest puste/zainicjowane (tylko typ Variant),
- `IsNumeric(w)` – określa czy `w` jest liczbą (pusta komórka jest liczbą),
- `IsObject(w)` – określa czy `w` reprezentuje obiekt.

Uwaga: funkcje **`Is`**... zwracają wartość logiczną **`True`**/**`False`**.

Określenie nazwy/numeru typu danych

- `TypeName(zmienna)` – zwraca nazwę typu zmiennej jako tekst,
- `VarType(zmienna)` – zwraca kod typu zmiennej.

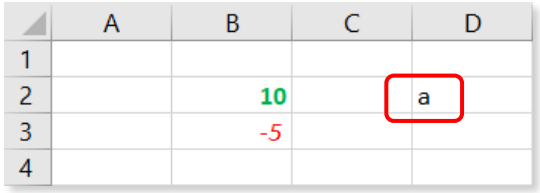
Kod	Nazwa
0	Empty
1	Null
2	Integer
3	Long
4	Single
5	Double
6	Currency
7	Date

Kod	Nazwa
8	String
9	Object
10	Error
11	Boolean
12	Variant
13	Object
14	Decimal
17	Byte

Przykład

Modyfikacja procedury `Koloruj2`. Dodatkowy warunek zapobiega modyfikacji ustawień czcionki, gdy wartość w komórce nie jest liczbą (warunek `.Vallue>=0` będzie sprawdzany tylko gdy komórka zawiera liczbę).

```
Public Sub Koloruj3()
  With ActiveCell
    If Not IsEmpty(.Value) And IsNumeric(.Value) Then
      If .Value >= 0 Then
        .Font.Color = vbGreen
        .Font.Bold = True
      Else
        .Font.Color = vbRed
        .Font.Italic = True
      End If
    End If
  End With
End Sub
```



Kod dostępny na stronie przedmiotu

Zagnieżdżanie instrukcji sterujących

Zagnieżdżenie instrukcji – umieszczenie pewnej instrukcji sterującej w zasięgu działania innej instrukcji sterującej.

Przykład

```
If warunek1 Then                    'pierwsza instr. warunkowa
  if warunek2 Then instrukcja1      'druga instr. warunkowa
Else
  If warunek3 Then                  'trzecia instr. warunkowa
    instrukcja2
  Else
    instrukcja3
  End If
End If
```

Druga instrukcja warunkowa zostanie wykonana jeżeli `warunek1` jest prawdziwy (wykonanie `instrukcji1` zależy od `warunku1` i `warunku2`)

Trzecia instrukcja warunkowa zostanie wykonana jeżeli `warunek1` jest fałszywy (wykonanie `instrukcji2` i `instrukcji3` zależy od `warunku1` i `warunku3`)

Wcięcie – dodatkowy odstęp od lewego marginesu wprowadzany dla zagnieżdżonej instrukcji. Nie wpływa na wykonanie programu, zwiększa czytelność kodu.

Przykład – MsgBox jako funkcja

MsgBox może być wywołany jako funkcja. W takim przypadku zwraca informację o naciśniętym przycisku i może być wykorzystany do komunikacji z użytkownikiem (szczegóły składni s.2-10).

```
MsgBox(prompt, buttons, title) As Integer
```

Dostępne przyciski (argument `buttons`): `vbOKOnly`, `vbOKCancel`, `vbYesNoCancel`, `vbYesNo`, `vbRetryCancel`, `vbAbortRetryIgnore`.

Zwracane wartości: `vbOK`, `vbCancel`, `vbAbort`, `vbRetry`, `vbIgnore`, `vbYes`, `vbNo`.

Przykład

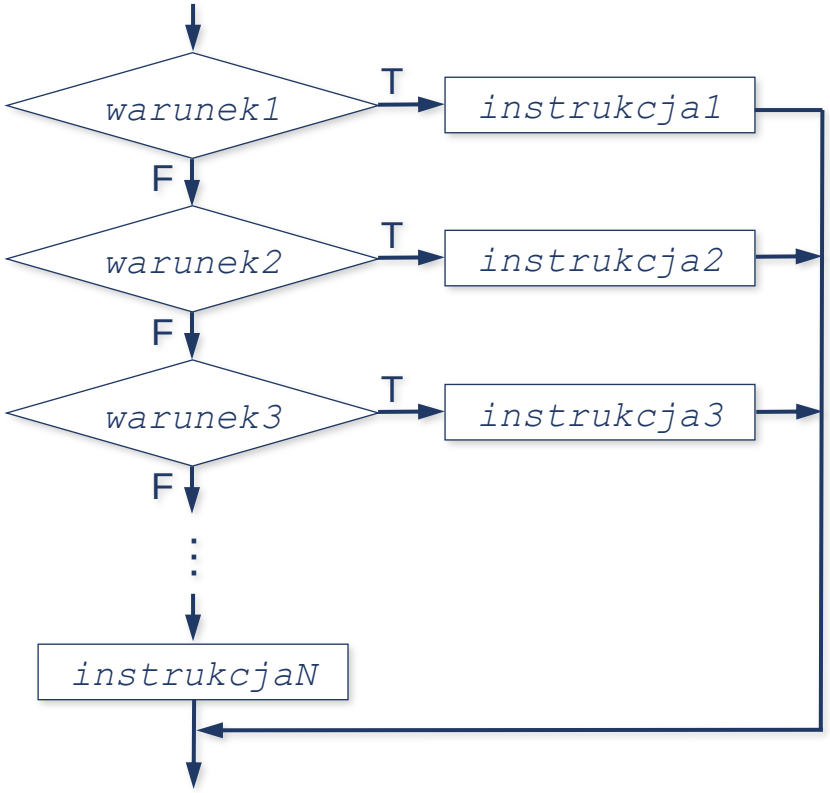
Procedura zmienia kolor tekstu w komórce C11 po akceptacji użytkownika

```
Public Sub ZmieńKolor()
  Dim odp As Integer
  odp = MsgBox("Zmienić kolor w C11?", vbQuestion + vbYesNo )
  If odp = vbYes Then Range("C11").Font.Color = vbRed
End Sub
```

Konstrukcja `ElseIf`

`ElseIf` jest opcjonalnym skladnikiem instrukcji **`If-End If`**. Może wystąpić wielokrotnie, każdy składnik wprowadza dodatkowy warunek.

```
If warunek1 Then
  instrukcja1-1
   …
  instrukcja1-N
ElseIf warunek2 Then
  instrukcja2-1
   …
  instrukcja2-N
ElseIf warunek3 Then
  instrukcja3-1
   …
  instrukcja3-N
ElseIf …
Else
  instrukcjaN-1
   …
  instrukcjaN-N
End If
```

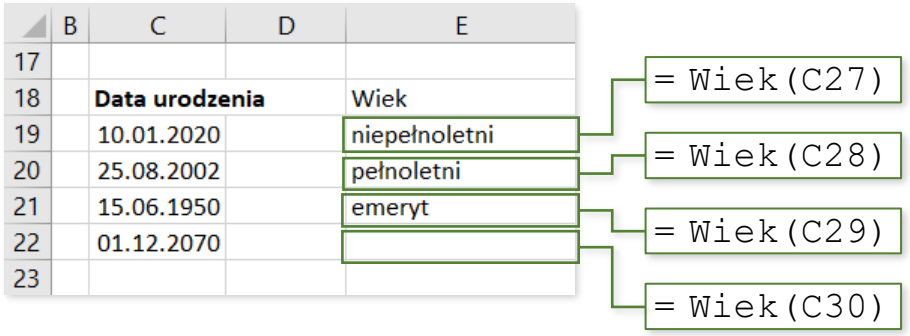


Uwaga: warunek *i+1* będzie sprawdzony jeżeli fałszywy był warunek *i-ty*. W każdym przebiegu może zostać wykonana co najwyżej jedna grupa instrukcji. Instrukcje w sekcji **`Else`** (opcjonalne) zostaną wykonane jeżeli wszystkie warunki były fałszywe.

Przykład – funkcja Wiek

Funkcja określa wiek na podstawie daty urodzenia (niepełnoletni, pełnoletni, emeryt).

```
Public Function Wiek(dataUr As Date) As String
  If dataUr > Date Then
    Wiek = ""
  Else
    If DateAdd("yyyy", 18, dataUr) > Date Then
      Wiek = "niepełnoletni"
    ElseIf DateAdd("yyyy", 65, dataUr) > Date Then
      Wiek = "pełnoletni"
    Else
      Wiek = "emeryt"
    End If
  End If
End Function
```



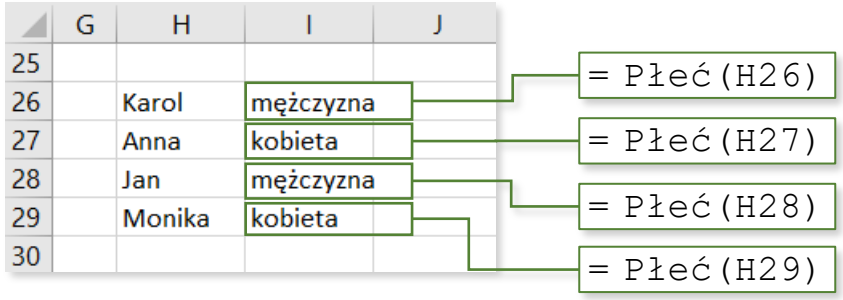
`Date` – funkcja zwraca aktualną datę

`DateAdd(`*`interwał`*`,`*`liczba`*`,`*`data`*`)` – dodaje do podanej *`daty`* *`interwał`* czasowy określony *`liczbą`* (możliwe interwały: `yyyy` – rok, `q` – kwartał, `m` – miesiąc, `d` – dzień, `ww`- tydzień, `h` – godzina, `n` – minuta, `s` – sekunda)

Przykład – funkcja Płeć

Funkcja określa płeć na podstawie imienia (działa dla imion polskich).

```
Public Function Płeć(imię As String) As String
  Dim znak As String
  znak = Right(imię, 1)
  If znak = "" Then
    Płeć = ""
  ElseIf znak = "a" Or znak = "A" Then
    Płeć = "kobieta"
  Else
    Płeć = "mężczyzna"
  End If
End Function
```



`Right(tekst,liczba)` – funkcja zwraca określoną `liczbę` znaków z prawej strony `tekstu` (końcowe znaki tekstu).

Kod dostępny na stronie przedmiotu