

# Microsoft Small Basic

*Въведение в програмирането*

Димитър Минчев & Мариан Панков

Настоящото учебно пособие е авторски и оригинален превод от Английски език на ръководството по програмиране на Small Basic на компанията Microsoft. Разрешава се разпространението на книгата, ако това е с нетърговска цел при изричното позоваване на източника и авторите. За издаване на материала на хартиен носител или продажбата на електронни копия трябва да имате договор с авторите.

© Димитър Минчев, Мариан Панков

**Microsoft Small Basic. Въведение в програмирането.**

Бургаски Свободен Университет, 2015

ISBN 978-619-7126-03-7

Настоящото учебно пособие е предназначено за всички студенти от първи курс по бакалавърски програми в Центъра по информатика и технически науки на Бургаски свободен университет изучаващи дисциплината „CS117 Основи на компютърната техника и програмирането“.

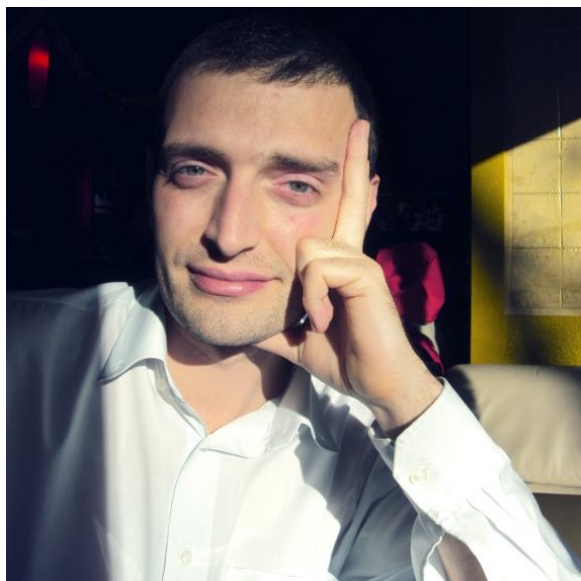
Всеки студент по тази дисциплина е необходимо да разработи семестриален проект, представляващ компютърна програма, реализираща интерактивна игра, разработена чрез средствата предоставяни от визуалната среда и езика за програмиране Small Basic на компанията Microsoft.

Реализираната от студента компютърна игра използва графичен режим и се управлява от потребителя посредством клавиатура и/или мишка, като студента има възможността сам да избира темата на играта.

Семестриалният проект трябва: да съдържа описание на реализираната от студента компютърна игра, да е подходящо оформен в текстов документ на Microsoft Office Word, да бъде представен като разпечатка на хартиен (листи формат A4) и електронен носител (диск).

Проекта се предава в деня на изпита по дисциплината, където студента демонстрира работата на разработената и описана в неговия проект програма в лаборатория компютърен клас на Бургаски свободен университет, с което той успешно защитава своята разработка и получава оценка.

### Димитър Минчев



Димитър Минчев е университетски преподавател в Център по информатика и технически науки на Бургаски свободен университет. Ръководител е на „Академията за таланти по програмиране“ школа по програмиране за ученици в Бургас.

Образователно квалификационни степени „Бакалавър“ и „Магистър“ по специалност „информатика“ получава съответно през 2007 от Бургаски свободен университет и през 2012 от Шуменски университет „Епископ Константин Преславски“. Образователна квалификационна степен „Доктор“ по специалност „Информатика“ получава през 2012 от Института по информационни и комуникационни технологии на Българската академия на науките.

Е-mail: [dimitar.minchev@gmail.com](mailto:dimitar.minchev@gmail.com)

### Мариан Панков



Мариан Панков получава образователна квалификационни степени „Бакалавър“ и „Магистър“ от университета "Проф. д-р Асен Златаров" – Бургас. Понастоящем работи като графичен дизайнер във вестник „Черноморски фар“ Бургас. В свободното си време е ИТ ентусиаст и любител програмист.

Е-mail: [marian.s.pankov@gmail.com](mailto:marian.s.pankov@gmail.com)

## Глава 1

# Въведение

---

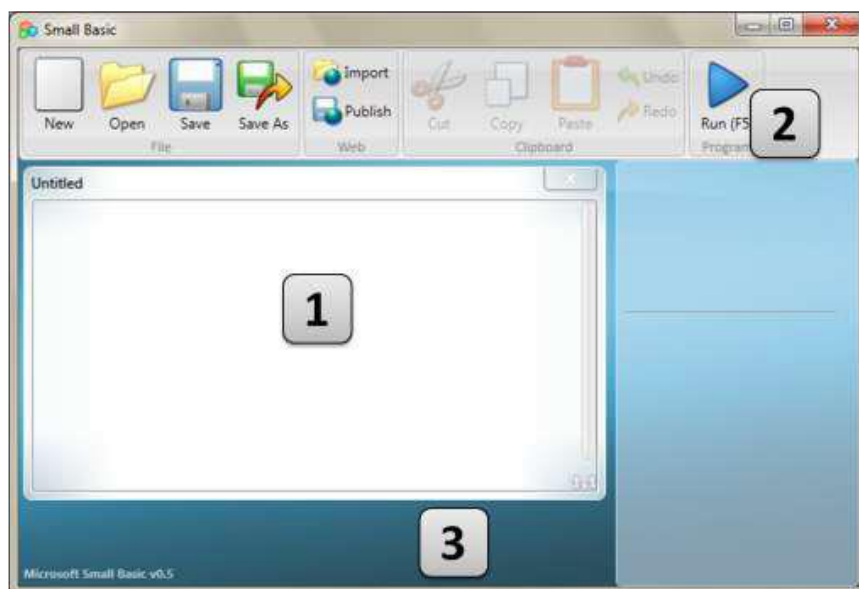
### Small Basic и програмиране

Компютърно програмиране се дефинира като процес на създаване на компютърен софтуер с използване на програмни езици. Както ние говорим и разбираме английски, испански или френски, компютрите могат да разбират програми написани на определени езици, наречени програмни. В началото е са съществували малко програмни езици, наистина лесни за заучаване и разбиране. Но след като компютрите и софтуера се усложняват повече и повече, програмните езици еволюират бързо, получавайки по пътя си на развитие по-комплексни концепции. В резултат на това повечето модерни програмни езици и техните концепции са сериозно предизвикателство за начинаещите. Този факт е обезкуражавал хората в изучаването и опита в компютърното програмиране.

Small Basic е програмен език, създаден да направи програмирането лесно достъпно и забавно за начинаещия. Целта на Small Basic е да сваля бариерата и да служи като началната стъпка към чудния свят на програмирането.

## Среда за програмиране

Нека започнем с бърз преглед на средата на Small Basic. Когато стартирате за първи път SmallBasic ще видите прозорец изглеждащ по следния начин:



Фигура 1: Средата за програмиране

Това е средата на Small Basic, където ние ще пишем и стартираме Small Basic програми. Тази среда има няколко отделни елемента обозначени с номера.

Editor, обозначен с (1), е мястото, където ще пишете Вашите програми. Когато отворите примерна програма или предварително записана програма, тя ще бъде изведена там. Редактор, в който се съдържа програмата, върху която работите в момента, се нарича активен.

Toolbar, обозначен с (2), се използва за извеждане на команди за активния редактор или за средата. В процеса на работа ще научим за различни команди.

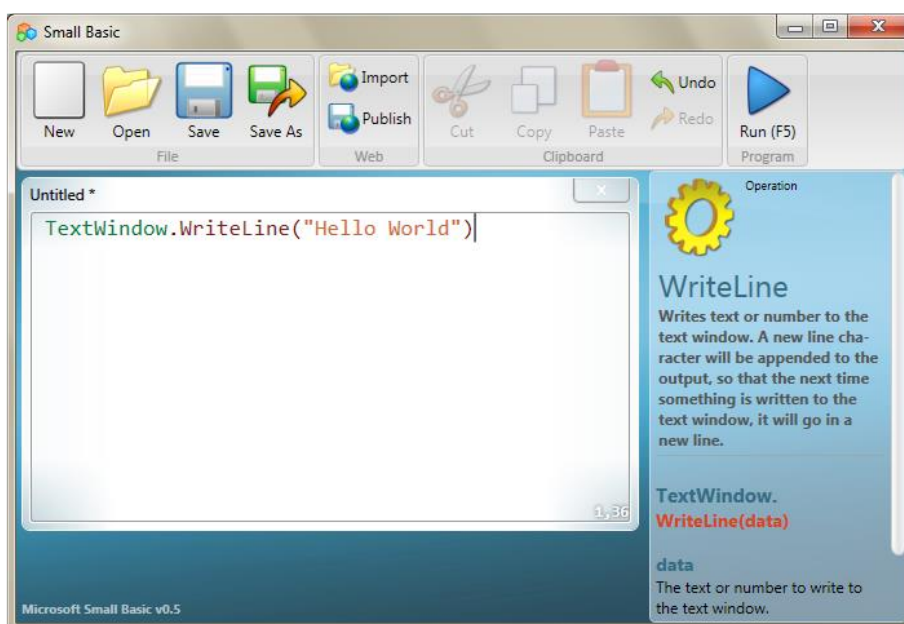
Surface, обозначен с (3), е мястото, където се намират всички прозорци на редактора.

## Нашата първа програма

Сега, когато сте запознати със средата на програмиране, ще продължим напред и ще започнем да програмираме. Както отбелязахме по-нагоре, редактор е мястото, където пишем нашите програми. Нека напишем следния ред там

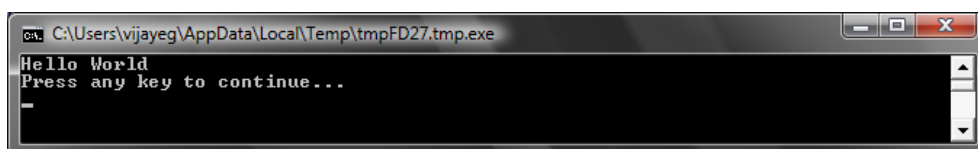
```
TextWindow.WriteLine("Hello World")
```

Това е Вашата първа програма на Small Basic. Ако сте я написали правилно, ще видите това, което е показано на следващата фигура.



Фигура 2: Първата програма

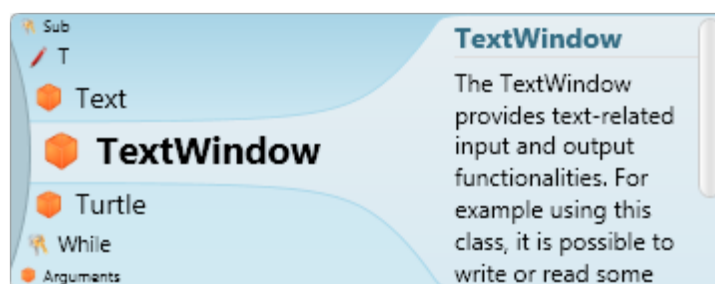
Сега след като написахме нашата първа програма нека я стартираме. Може да го направим или като щракнем на бутон RUN, или с клавиш от клавиатурата – F5. Ако всичко върви добре, нашата програма ще се стартира и резултатът е показан на фигурата по-долу.



Фигура 3: Изход на първата програма

Поздравления! Вие току що написахте и стартирахте вашата първа програма на Small Basic. Много малка и лесна програма, но въпреки това голяма крачка напред към превръщането Ви в истински компютърен програмист. Сега има само един детайл, преди да създадем по-голяма програма. Трябва да разберем какво точно се случи – това, което в същност казахме на компютъра и как той разбра, какво трябва да направи? За да добием представа, в следващата глава ще анализираме програмата, която току що написахме.

*Пишейки Вашата първа програма, може би забелязахте появата на прозорец във формата на списък. Той помага за по-бързо писане при програмирането. Може да преминете през списъка натискайки стрелките Up/Down, и като намерите това което желаете може да натиснете клавиша Enter, за да го вмъкнете във вашия програмен код.*



Фигура 4:

## Записване на нашата програма

Ако желаете да затворите Small Basic и да продължите работа върху програмата по-късно можете да я запишете. Всъщност е добра практика да се записват програмите от време на време, за да не загубите информация в случай на случайно изключване на компютъра или срыв в храненето. Може да запишете програмата както с щракане върху иконата SAVE на тулбара, така и с клавишна комбинация „Ctrl+S“ (натискате клавиш S докато задържате клавиш Ctrl).

# Разбиране на нашата първа програма

---

## Какво е всъщност компютърна програма?

Програма е съвкупност от инструкции за компютъра. Тези инструкции казват на компютъра точно какво да прави и той винаги ги следва. Като хората, компютрите могат да следват инструкции, които са написани на език, който компютрите могат да разберат. Тези езици се наричат програмни. Съществуват много такива и Small Basic е един от тях.

Представете си разговор между Вас и ваш приятел. За да пренесете информация между Вас, Вие и вашите приятели използвате думи, организирани в изречения. По сходен начин програмните езици съдържат колекции от думи, които могат да бъдат организирани в изречения, които да пренасят информацията към компютъра. Програмите най-просто казано са съвкупност от изречения (понякога малко на брой, понякога хиляди), заедно имащи значение, както за програмиста така и за компютъра.

*Има много езици, които компютърът може да разбира. Java, C++, Python, VB и т.н. са мощни, модерни компютърни езици, използвани за разработка на прости или комплексни софтуерни програми.*

## Програми на Small Basic

Типичната Small Basic програма се състои от група изречения. Всеки ред от програмния код представя израз и всяко израз е инструкция за компютъра. Когато поискаме от компютъра да изпълни програма, той я взима и прочита първото израз. Разбира какво се опитваме да кажем, след което изпълнява нашата инструкция. Когато приключи изпълнението на първият израз, той се връща към програмата и изпълнява втория ред. Продължава така докато достигне края на програмата. Тогава и нашата програма приключва.

## Да се върнем към нашата първа програма

Това е първата програма, която написахме:

```
TextWindow.WriteLine("Hello World")
```

Това е много проста програма, състояща се от един израз. Тя казва на компютъра да напише ред от текст – Hello World, в текстовия прозорец.

Буквално преведено от компютъра в смисъл на:

```
Write Hello World
```

Може би сте забелязали, че изразът може да бъде разделен на по-малки сегменти, също както изречението на думи. В първия израз имаме 3 отделни сегмента:

**a) TextWindow**

**b) WriteLine**

**c) "Hello World"**

Точките, скобите и кавичките са пунктуационни знаци, които е необходимо да бъдат поставени на необходимите места в израза, така че компютъра да разбере нашата цел.

Може би си спомняте черния прозорец, който се появи когато стартирахме нашата първа програма. Черният прозорец се нарича TextWindow, или също така конзола. Там се извежда и резултата от програмата. TextWindow в наша-

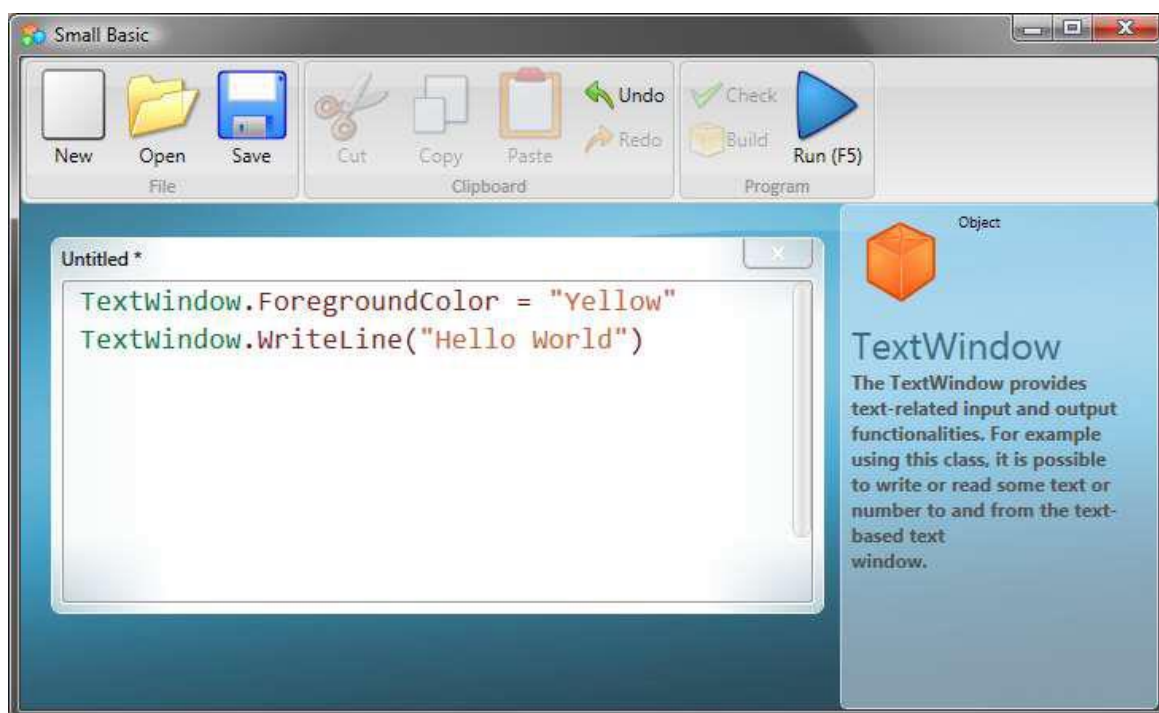
та програма се нарича *обект*. Съществуват голям брой такива обекти, достъпни за нас, за използване в нашите програми. Може да изпълняваме няколко различни *операции* върху тези обекти. Вече сме използвали операцията *WriteLine* в нашата програма. Забелязали сте също, че операцията *WriteLine* е последвана от *Hello World* в кавички. Този текст преминава през входа на операцията *WriteLine*, която в последствие го извежда на екран за потребителя. Това се нарича *вход* за операцията. Някои имат един или няколко, докато други нямат нито един.

Пунктуационни знаци като кавички, интервали и скоби са много важни за компютърната програма. В зависимост от мястото и броя им, те могат да променят смисъла на това което се очаква.

## Нашата втора програма

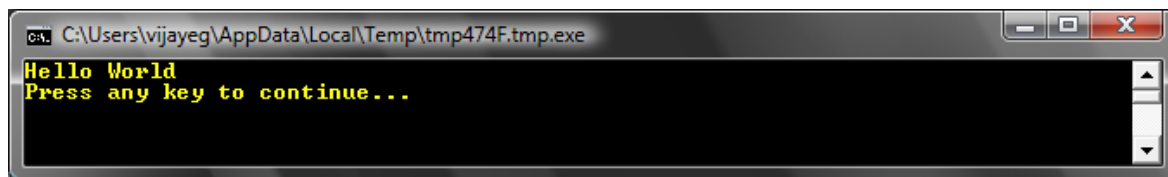
Вече сте разбрали нашата първа програма и нека продължим напред и я направим по-пъстра, добавяйки няколко цвята.

```
TextWindow.ForegroundColor = "Yellow"  
TextWindow.WriteLine("Hello World")
```



Фигура 5: Добавяне на цветове

Когато изпълните програмата ще забележите, че тя отново извежда изрaza „Hello World“ в конзола, но този път в жълто, а не в сиво както в предишния случай.



Фигура 6: "Hello world" в жълто

Обърнете внимание на новия израз, който добавихме към оригиналната програма. Той използва нова дума `ForegroundColor`, която има стойност „Yellow“. Това означава че сме определили „Yellow“ за *ForegroundColor*. Разликата между двете е в това че *ForegroundColor* няма входове и няма нужда от скоби. Вместо това е последвана от знак за равенство към символ и дума. Дефинираме *ForegroundColor* като свойство на `TextWindow`. Вижте списъка със стойности валидни за свойството *ForegroundColor*. Опитайте да заместите „Yellow“ с една от тях и вижте резултата. Не забравяйте кавичките, те са задължителни пунктуационни знаци.

Black  
Blue  
Cyan  
Gray  
Green  
Magenta  
Red  
White  
Yellow  
DarkBlue  
DarkCyan  
DarkGray  
DarkGreen  
DarkMagenta  
DarkRed  
DarkYellow

## Представяне на променливи

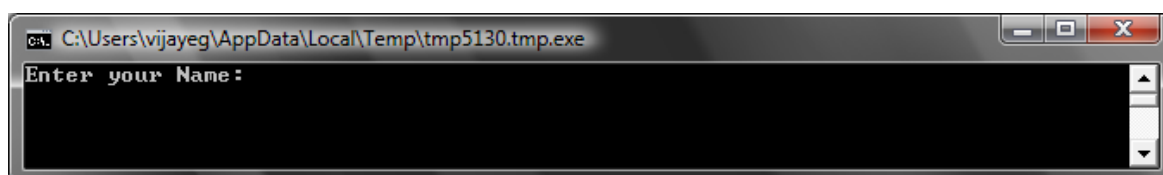
---

### Използване на променливи в нашата програма

Не би ли било прекрасно, ако нашата програма може да изпише „Hello“ с името на потребителя вместо обикновеното „Hello World“. За да постигнем това първо трябва да попитаме потребителя за името му, след което да го запазим някъде и да изведем на екран „Hello“ и това име. Да видим как ще го направим:

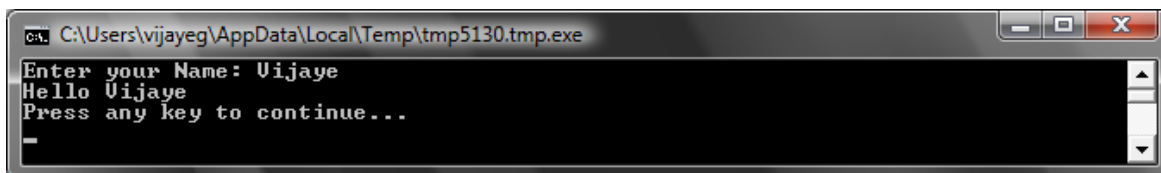
```
TextWindow.Write("Enter your Name: ")
name = TextWindow.Read()
TextWindow.WriteLine("Hello " + name)
```

Когато напишете и изпълните програмата ще видите резултат като следващия:



Фигура 7: Изискване името на потребителя

И когато напишете името и натиснете Enter, ще видите следното:



Фигура 8: Резултатът

Ако стартирате програмата отново ще ви бъде зададен същия въпрос. Може да изпишете друго име и компютърът ще изпише „Hello“ с това име.

## Анализ на програмата

Редът в програмата, която току що стартирахте, който сигурно ще ви направи впечатление е следния:

```
name = TextWindow.Read()
```

*Read()* изглежда почти като *WriteLine()*, но без входове. Това е операция и казва на компютъра да изчака потребителя да въведе нещо и да натисне клавиш Enter. Когато потребителя натисне Enter, компютърът взима написаното и го връща в програмата. В случая интересното е, че което е написано от клавиатурата е запазено в променлива наречена *name*. Променлива се дефинира като място, където стойности могат да бъдат запазвани и използвани по-късно. В реда по-горе, *name* бе използвано за съхранение на името на потребителя.

Интересен е и следния ред:

```
TextWindow.WriteLine("Hello " + name)
```

Това е мястото, където използваме стойността съхранена в нашата променлива – *name*. вземаме стойността в *name*, прикачваме я към „Hello“ и я извеждаме в конзолата.

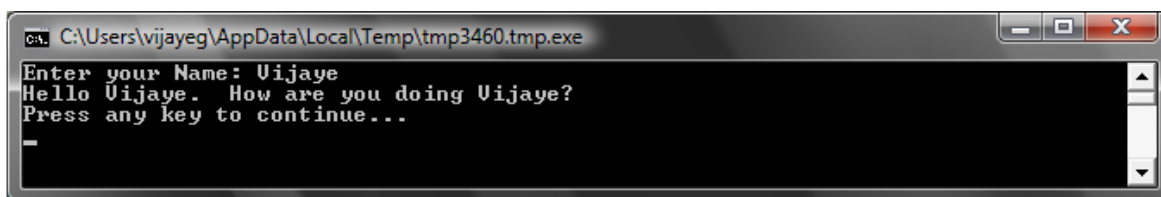
Веднъж зададена променлива може да бъде използвана неограничен брой пъти. Например може да направите следното:

```
TextWindow.Write("Enter your Name: ")
```

*Write също като WriteLine е още една операция за TextWindow. Тя ви позволява да изпишете нещо на TextWindow, но предполага следващия текст да бъде на същия ред на който е и текущия.*

```
name = TextWindow.Read()  
TextWindow.Write("Hello " + name + ".")  
TextWindow.WriteLine("How are you doing " + name + "?")
```

Ще видите следния резултат:



Фигура 9: Повторно използване на променлива

## Правила за наименоване на променливи

Променливите имат свързани с тях имена, които помагат за идентифицирането им. Съществуват няколко прости правила и добри съвети за именуване на променливи. Те са:

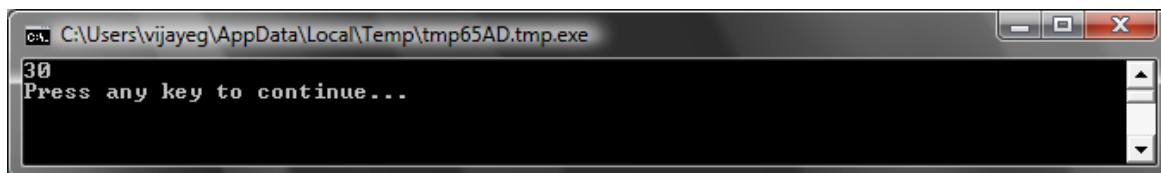
1. Името винаги трябва да започва с буква и да не си противоречи с ключови думи като if, for, then и т.н.
2. Името може да съдържа комбинация от букви, цифри и долни черти.
3. Полезно е променливите да са именувани смислово т.е. след като променливите могат да бъдат неограничено дълги, използвайте имената им за описание на целта им.

## Експериментиране с числа

Вече разбрахме как може да се използват променливи за съхранение на името на потребителя. В следващите няколко програми ще видите как могат да се съхраняват и манипулират числа в променливи. Нека започнем с наистина лесна програма:

```
number1 = 10  
number2 = 20  
number3 = number1 + number2  
TextWindow.WriteLine(number3)
```

Когато изпълните програмата ще видите следния резултат:



Фигура 10: Събиране на две числа

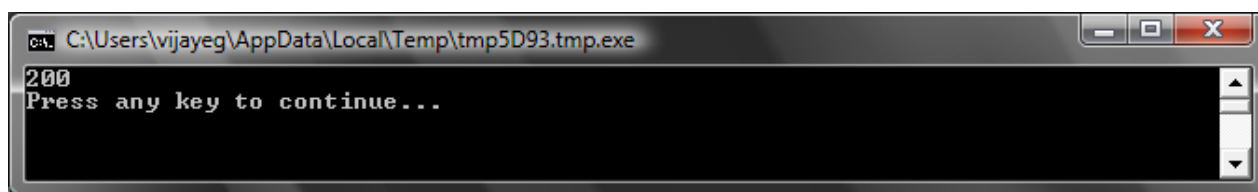
В първия ред на програмата приписват стойност 10 на променлива number1. Във втория ред съответно – 20 на number2. В третия ред събирате number1 и number2, след което приписвате резултата на number3. Така че в този случай number3 ще има стойност 30. И това извеждаме на екран.

*Забележете, че числата нямат кавички. За числа кавички не са необходими. Използвайте ги само когато работите с текст*

Нека променим малко програмата и видим резултата:

```
number1 = 10
number2 = 20
number3 = number1 * number2
TextWindow.WriteLine(number3)
```

Програмата по-горе ще умножи number1 и number2 и ще съхрани резултата в number3. Може да видите резултата по-долу:



Фигура 11: Умножение на две числа

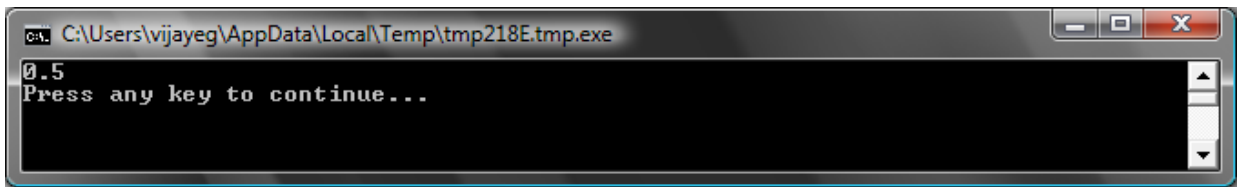
Аналогично може да изваждате или делите числа. Изваждане:

```
number3 = number1 - number2
```

Със символа за делене „/“ програмата ще изглежда така:

```
number3 = number1 / number2
```

Резултатът от делението би бил:



Фигура 12: Делене на две числа

## Прост конвертор за температура

За следващата програма ще използваме формулата  $^{\circ}\text{C} = \frac{5(^{\circ}\text{F}-32)}{9}$  за конвертиране на температура от Фаренхайт към Целзий.

Първо ще вземем температурата във Фаренхайт от потребителя, след което ще я съхраним в променлива. Операцията, която чете числа от потребителя е **TextWindow.ReadNumber**.

```
TextWindow.Write("Enter temperature in Fahrenheit: ")  
fahr = TextWindow.ReadNumber()
```

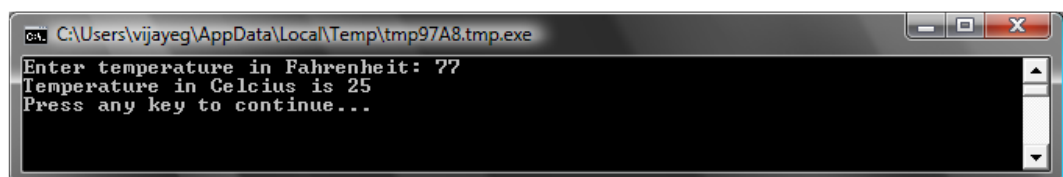
След като имаме температурата в променлива може да я конвертираме в Целзий по следния начин:

```
celsius = 5 * (fahr - 32) / 9
```

Скобите показват на компютъра кое трябва да изчисли първо преди останалите операции. Остава да изведем на екран резултата за потребителя. Цялата програма изглежда така:

```
TextWindow.Write("Enter temperature in Fahrenheit: ")  
fahr = TextWindow.ReadNumber()  
celsius = 5 * (fahr - 32) / 9  
TextWindow.WriteLine("Temperature in Celsius is " + celsius)
```

И резултатът от нея би бил:



Фигура 13: Конвертор за температура

## Условия и разклонения

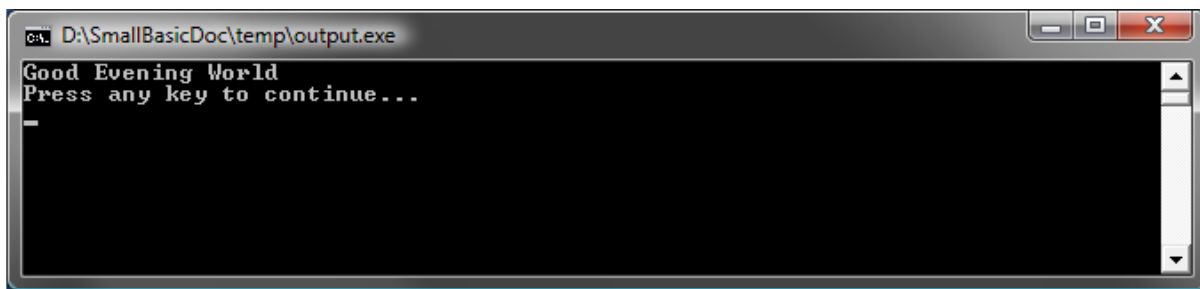
Не би ли било хубаво, ако с нашата първа програма не казваме обикновеното „Hello World“, а вместо това казваме „Good Morning World“ или „Good Evening World“ според времето от деня. Със следващата програма ще накараме компютъра да изпише *Good Morning World*, ако часът е преди 12 през деня, или *Good Evening*, ако е след този час.

```
If (Clock.Hour < 12) Then
    TextWindow.WriteLine("Good Morning World")
EndIf
If (Clock.Hour >= 12) Then
    TextWindow.WriteLine("Good Evening World")
EndIf
```

Според времето, в което изпълнявате програмата ще видите един от двата резултата :



Фигура 14: Good Morning World



Фигура 15: Good Evening World

Нека анализираме първите три реда от програмата. Вероятно вече сте разбрали, че този ред казва на компютъра да изпише „Good Morning World“, ако `Clock.Hour` е по-малко от 12. Думите **If**, **Then**, **Endif** са специални и разбираеми за компютъра, когато програмата се изпълнява. Думата **If** винаги е следвана от условие, което в случая в `(Clock.Hour < 12)`. Запомнете, че скобите са необходими с цел компютъра да разбере Вашите намерения. Условието е следвано от **Then** за да се изпълни текущата операция. След операцията идва **Endif**. Това казва на компютъра, че условието е изпълнено.

*В Small Basic може да използвате обекта `Clock` за да получите достъп до текущите дата и час. Той също предлага и група от свойства, които ви позволяват да получите текущите ден, месец, година, час, минута, секунда отделно.*

Между **then** и **endif** може да има повече от една операция и компютъра ще ги изпълни всички, ако условието е спазено. Например може да напишете нещо като това:

```
If (Clock.Hour < 12) Then
    TextWindow.Write("Good Morning. ")
    TextWindow.WriteLine("How was breakfast?")
EndIf
```

## Else

Забелязали сте, че второто условие в програмата представена в началото на тази глава е излишно. Стойността на **Clock.Hour** може да бъде или да не бъде по малко от 12. В действителност не бе необходимо да правим втората про-

верка. В случаи като този изразите **If...Then...Endif** може да бъдат съкратени до един с използването на нова дума – **Else**.

Ако пренапишем програмата използвайки тази дума, това ще изглежда така:

```
If (Clock.Hour < 12) Then  
    TextWindow.WriteLine("Good Morning World")  
Else  
    TextWindow.WriteLine("Good Evening World")  
EndIf
```

Тази програма ще изпълни същото както и предходната, което ни довежда до много важен урок в програмирането:

“ В програмирането обикновено има много начини за изпълнение на едно и също нещо. Понякога един начин има повече смисъл от друг. Изборът остава на програмиста. Пишейки повече програми и добивайки повече опит, вие ще започнете да забелязвате тези различни техники и да откривате качествата и недостатъците им.

## Новия ред

Във всички примери виждате как изразите между **if**, **else** и **endif** са на нов ред. Това подреждане не е необходимо. И без него компютърът ще разбере програмата. Новите редове, обаче, ни помагат да разберем по-лесно структурата на програмата. Следователно е добра практика изразите между такива блокове да се разполагат на нов ред.

## Четно и нечетно

След като имаме на разположение изразите **If..Then..Else..EndIf**, нека напишем програма, която при дадено число може да ни каже дали това число е четно или нечетно.

```
TextWindow.Write("Enter a number: ")  
num = TextWindow.ReadNumber()
```

```
remainder = Math.Remainder(num, 2)
If (remainder = 0) Then
    TextWindow.WriteLine("The number is Even")
Else
    TextWindow.WriteLine("The number is Odd")
EndIf
```

Изпълнената програма е с резултат:



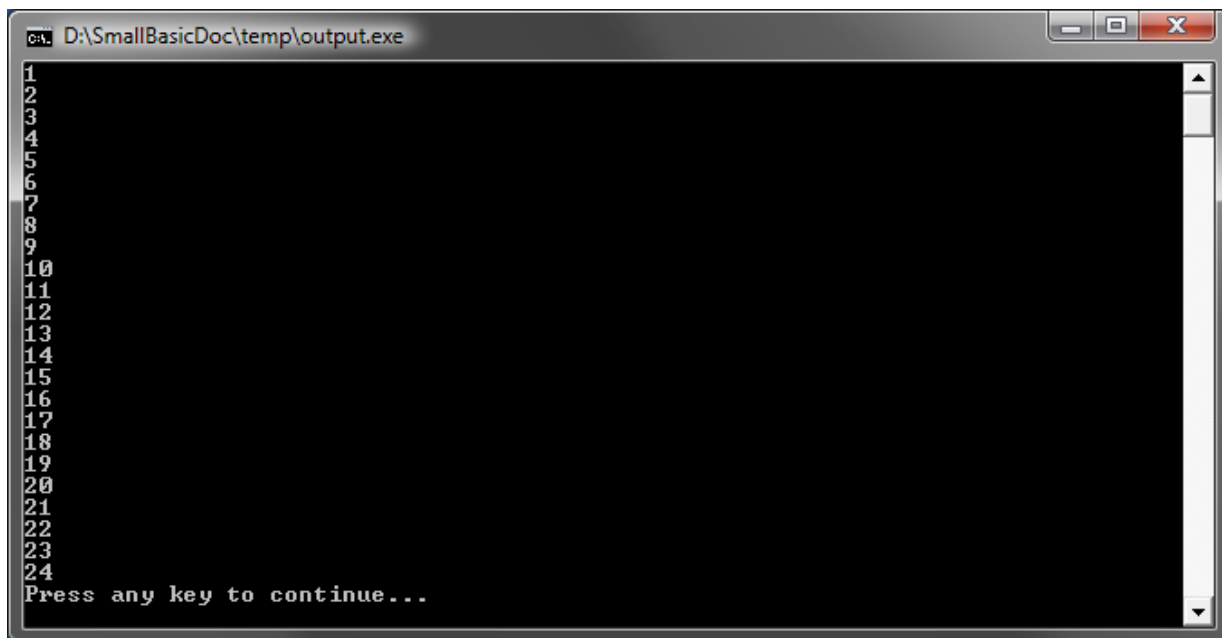
Фигура 16: Четни и нечетни

Тук сме представили една нова полезна операция `Math.Remainder`. Може би сте разбрали, че `Math.Remainder` ще раздели първото с второто число и ще върне остатъка.

## Разклонения

Във втора глава научихте, че компютърът изпълнява програмата последователно по един израз в последователност отгоре надолу. Съществува обаче специален израз, който може да го накара да прескочи на друг израз извън този ред. Нека да разгледаме следващата програма:

```
i = 1
start:
TextWindow.WriteLine(i)
i = i + 1
If (i < 25) Then
    Goto start
EndIf
```



Фигура 17: Използване на Goto

В тази програма приписваме стойност 1 на променлива *i*. След това добавяме израз завършващ с двоеточие (:

```
start:
```

Това се нарича етикет. Етикетите са като отметки, които компютърът може да разбере. Може да наименоувате като отметка всичко и може да поставяте колкото желаете етикети в програмата, стига те да имат уникални имена.

Още един интересен израз е и:

```
i = i + 1
```

Този ред казва на компютъра да добави 1 към променливата *i*, след което резултатът се приписва отново на *i*. Следователно ако *i* е била 1 преди израза, ще бъде 2 след изпълнението му.

И накрая:

```
If (i < 25) Then  
Goto start  
EndIf
```

Тази част казва на компютъра – ако стойността на *i* е по-малка от 25 да се изпълни израза след отметката *start*.

## Безкрайно изпълняване

Използвайки израза Goto можете да накарате компютъра да повтори нещо няколко пъти. Например програмата за четни и нечетни числа, преработена по начина по-долу, ще се изпълнява вечно. Може да я спрете с бутона X в горния десен ъгъл на прозореца.

```
begin:
  TextWindow.Write("Enter a number: ")
  num = TextWindow.ReadNumber()
  remainder = Math.Remainder(num, 2)
  If (remainder = 0) Then
    TextWindow.WriteLine("The number is Even")
  Else
    TextWindow.WriteLine("The number is Odd")
  EndIf
  Goto begin
```



Фигура 18: Програма за четни и нечетни числа, който се изпълнява безкрайно

## Цикъл For

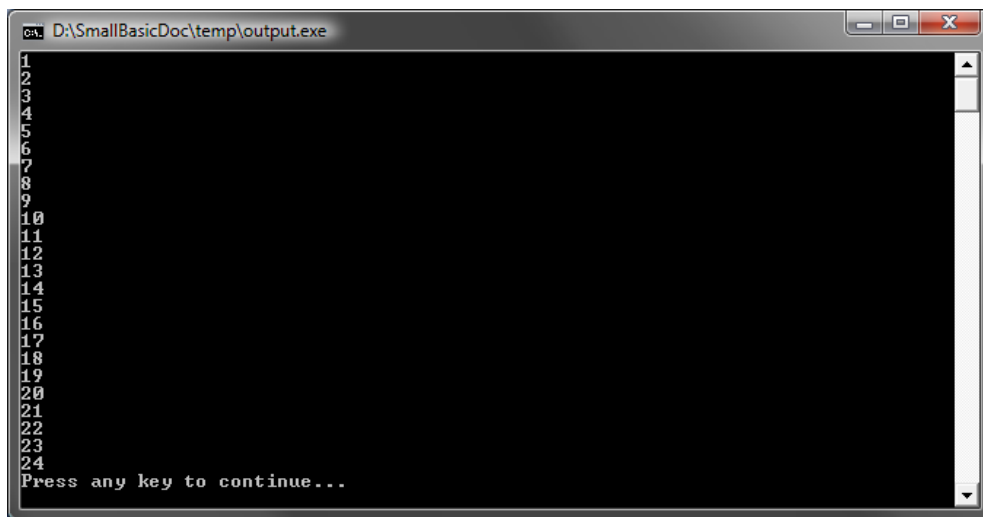
Да погледнем програмата, която написахме в последната глава.

```
i = 1
start:
TextWindow.WriteLine(i)
i = i + 1
If (i < 25) Then
    Goto start
EndIf
```

Тя извежда на екран последователно числата от 1 до 24. Процесът на нарастване на променливата е често срещан в програмирането, за това и програмните езици предлагат лесни методи за изпълнението му. Предишната и следващата програма са идентични:

```
For i = 1 To 24
    TextWindow.WriteLine(i)
EndFor
```

И резултатът е:



Фигура 19: Използване на цикъл For

Забележете, че намалихме броя на редовете от 8 на 4 и програмата от 4 реда прави същото, което и тази от 8 реда. Помнете, че по-рано споменахме, че обикновено има няколко начина да се изпълни едно и също нещо. Това е чудесен пример:

**For ... EndFor** като термин в програмирането се нарича цикъл. Той позволява на компютъра, след зададена начална и крайна стойност на променлива, да извърши нарастването ѝ. Всеки път когато променливата нараства се изпълнява израз между **For** и **EndFor**.

Ако желаете вместо с 1 променливата да нараства с 2, след което да изведете на екран всички нечетни числа от 1 до 24, можете да го направите така:

```
For i = 1 To 24 Step 2
    TextWindow.WriteLine(i)
EndFor
```



Фигура 20: Само нечетни числа

Частта от израза **For – Step 2** казва на компютъра да извърши нарастването на *i* с 2 вместо с обичайното – 1. Използвайки **Step** може да задавате каквото пожелаете нарастване. За стъпка можете да задавате и отрицателни стойности като по този начина карате компютъра да изчислява стойностите в обратен ред, както в примера:

```
For i = 10 To 1 Step -1  
    TextWindow.WriteLine(i)  
EndFor
```

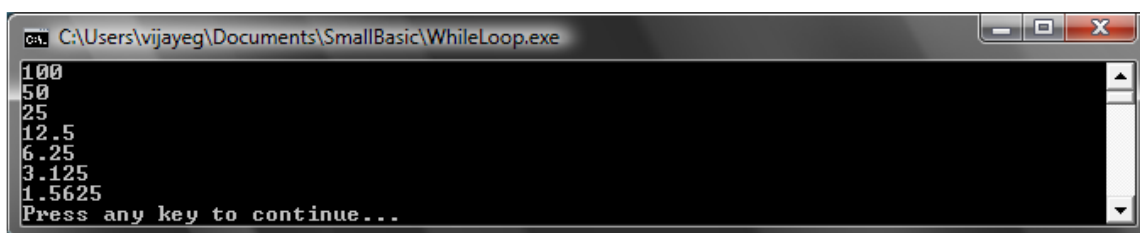


Фигура 21: Обратно броене

## While

Още един метод за създаване на цикъл е While. Той е полезен когато броя на циклите е неизвестен. Докато цикъла For се изпълнява определен брой пъти, While – до достигане на определено условие. В следващия пример получаваме число, докато това число е по-голямо от 1:

```
number = 100  
While (number > 1)  
    TextWindow.WriteLine(number)  
    number = number / 2  
EndWhile
```



Фигура 22: Разполовяване на цикъл

В тази програма на *number* приписване стойност 100 и изпълняваме цикъла While , докато полученото число е по-голямо от 1. Вътре в цикъла извеждаме числото на екран и го делим на 2. Очаквано е, че резултатът е редица от числа, която прогресивно намалява.

Наистина би било много трудно да напишем програмата с цикъл For, защото не знаем колко пъти е необходимо да бъде изпълнен той. С While е лесно да проверим дали е изпълнено определено условие и да накараме компютъра да продължи изпълнението на програмата или да я прекрати според това условие. Интересно е да се отбележи, че този цикъл може да бъде заместен с изрази If ... Then. Например програмата може да бъде пренаписана по следния начин, без да се промени крайния резултат:

```
number = 100
startLabel:
TextWindow.WriteLine(number)
number = number / 2

If (number > 1) Then
    Goto startLabel
EndIf
```

*Всъщност компютърът вътрешно пренаписва всеки цикъл While в If..Then изрази, както и един или няколко Goto изрази.*

# Графика за начинаещи

---

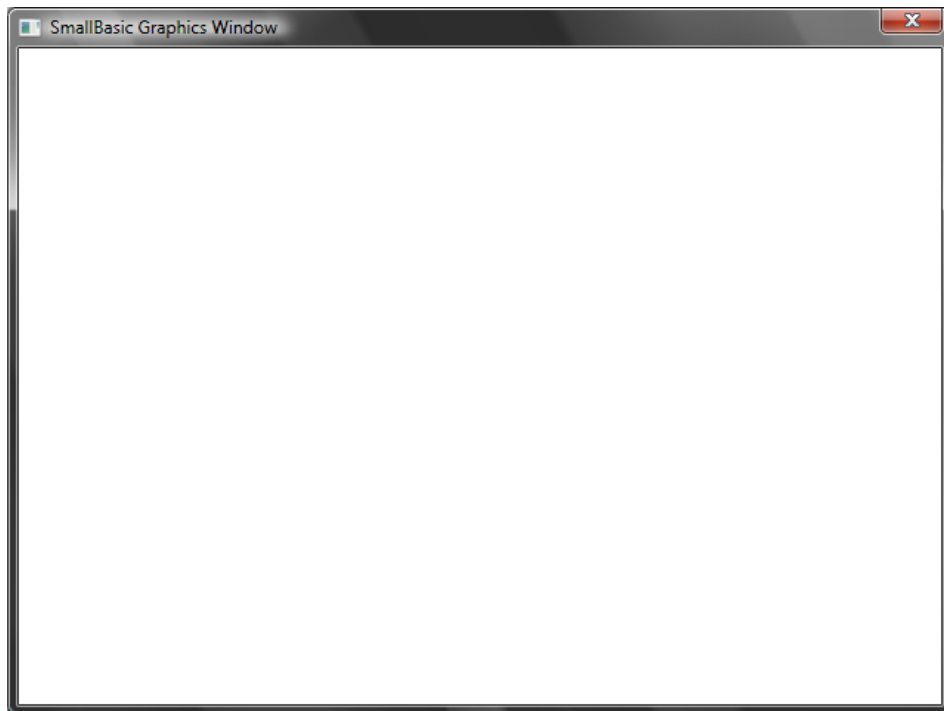
Дотук в нашите примери използвахме `TextWindow` за да обясним основите на `Small Basic`. Той обаче идва и с мощна група от графични инструменти, които ще разгледаме в следващата глава.

## Представяне на `GraphicsWindow`.

Ние използвахме `TextWindow` за да работим с текст и числа, но `Small Basic` предлага **`GraphicsWindow`**, позволяващ ни да рисуваме неща. Нека започнем неговото представяне:

```
GraphicsWindow.Show()
```

Когато стартирате програмата ще забележите, че вместо обичайния черен прозорец за текст, имате бял, като този показан по-долу. Няма почти нищо, което да правите с него все още. Той ще бъде основния прозорец, в който ще работим в тази глава. Може да затворите прозореца като натиснете бутона „X“ в горния десен ъгъл.



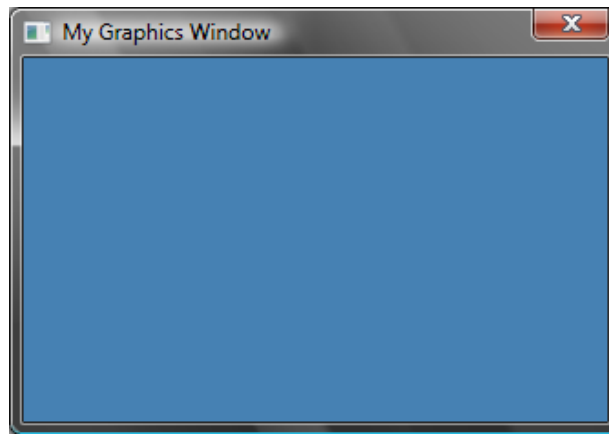
Фигура 23: Празен прозорец за графика

## Настройки на GraphicsWindow

Може да промените изгледа на прозореца както желаете: заглавието, фона или размерите му. Нека да го модифицираме малко, за да се запознаем по-добре с него.

```
GraphicsWindow.BackgroundColor = "SteelBlue"  
GraphicsWindow.Title = "My Graphics Window"  
GraphicsWindow.Width = 320  
GraphicsWindow.Height = 200  
GraphicsWindow.Show()
```

Ето как изглеждат промените. Може да промените фона с една от многото стойности, изведени в приложение В. Експериментирайте с тези свойства за да разберете как може да модифицирате прозореца.

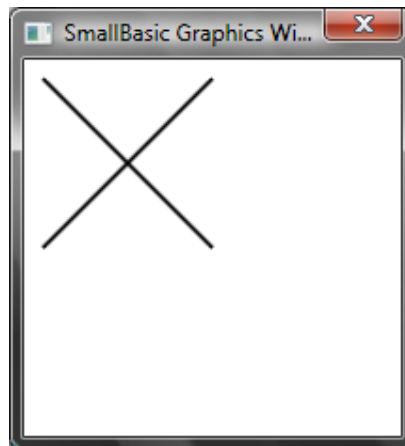


Фигура 24: Прозорец за графика променен според предпочитанията на потребителя

## Изчертаване на линии

Когато GraphicsWindow приготвен, върху него можете да рисувате фигури, текст и дори картини. Нека започнем с чертаенето на прости фигури. Това е програма, която рисува няколко линии на GraphicsWindow.

```
GraphicsWindow.Width = 200  
GraphicsWindow.Height = 200  
GraphicsWindow.DrawLine(10, 10, 100, 100)  
GraphicsWindow.DrawLine(10, 100, 100, 10)
```

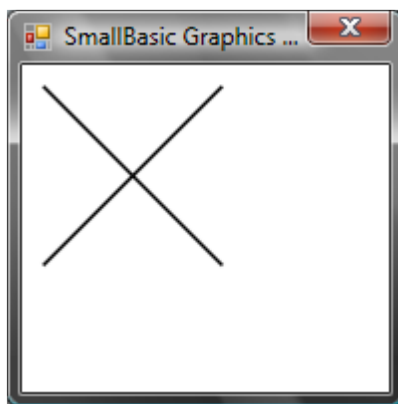


Фигура 25: Пресечени линии

Първите два реда от програмата настройват прозореца, а вторите два очертават пресечени линии. Първите две числа след *DrawLine* определят началните координати

*Вместо имена на цветове можете да използвате уеб нотация (#RRGGBB). Например #FF0000 означава Червено, #FFFF00 е Жълто и т.н. Ще научим повече за цветовете малко по късно.*

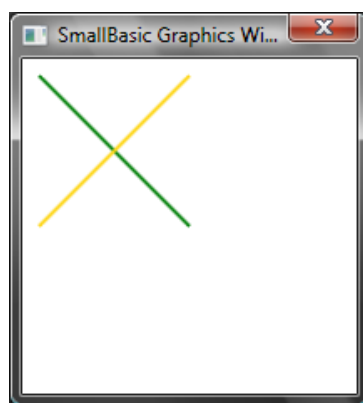
х и у, вторите две – крайните координати. Интересното свързано с компютърната графика е, че координатите (0,0) се намират в горния ляв ъгъл на прозореца. Следователно прозореца се счита за втория квадрант на координатната система.



Фигура 26: Координатна система

Да се върнем на програмата. Small Basic дава възможност да се променят свойствата на линиите, като цвят и дебелина. За начало нека променим цвета на линиите, както е показано в програмата по-долу.

```
GraphicsWindow.Width = 200  
GraphicsWindow.Height = 200  
GraphicsWindow.PenColor = "Green"  
GraphicsWindow.DrawLine(10, 10, 100, 100)  
GraphicsWindow.PenColor = "Gold"  
GraphicsWindow.DrawLine(10, 100, 100, 10)
```

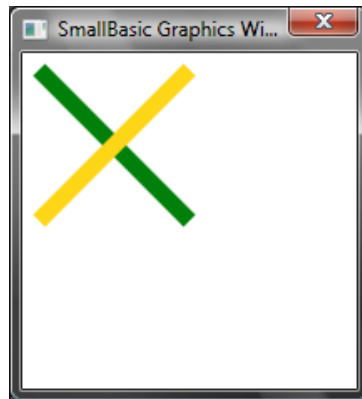


Фигура 27: Промяна цвета на линиите

Нека променим и размерите. В показаната програма променяме дебелината от 1 (което е стойността по подразбиране) на 10.

```
GraphicsWindow.Width = 200  
GraphicsWindow.Height = 200
```

```
GraphicsWindow.PenWidth = 10  
GraphicsWindow.PenColor = "Green"  
GraphicsWindow.DrawLine(10, 10, 100, 100)  
GraphicsWindow.PenColor = "Gold"  
GraphicsWindow.DrawLine(10, 100, 100, 10)
```

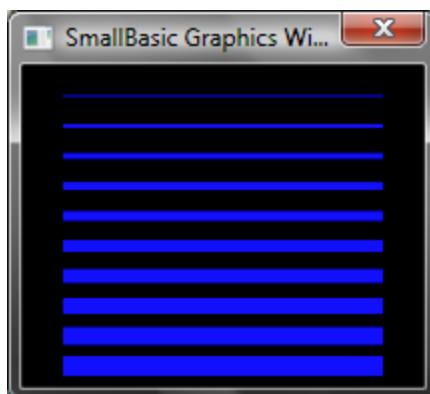


Фигура 28: Удебелени цветни линии

PenWidth и PenColor модифицират писецът, с който линиите са начертани. Те не засягат само линиите, но и фигурите, начертани след обновяване на свойствата.

С използване на изрази за цикъл може лесно на напишем програма, чертаеща няколко линии с повишаване на дебелината на писеца.

```
GraphicsWindow.BackgroundColor = "Black"  
GraphicsWindow.Width = 200  
GraphicsWindow.Height = 160  
GraphicsWindow.PenColor = "Blue"  
For i = 1 To 10  
    GraphicsWindow.PenWidth = i  
    GraphicsWindow.DrawLine(20, i * 15, 180, i * 15)  
Endfor
```

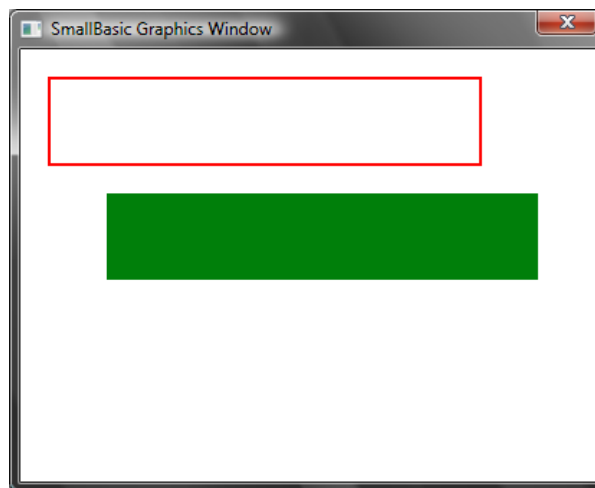


Интересната част от тази програма е цикъла, в който се повишава дебелината на писеца с всяко негово изпълнение, след което се изчертава нова линия под старата.

## Изчертаване и запълване на фигури.

Когато става въпрос за рисуване на форми, обикновено има два типа операции са всяка една от тях. Това са операциите *Draw* и *Fill*. Първата очертава контурите с използване на писец, втората рисува фигурата с използване на четка. На пример в следващата програма два правоъгълника са начертани, единият с червен молив и друг запълнен със зелена четка.

```
GraphicsWindow.Width = 400  
GraphicsWindow.Height = 300  
  
GraphicsWindow.PenColor = "Red"  
GraphicsWindow.DrawRectangle(20, 20, 300, 60)  
  
GraphicsWindow.BrushColor = "Green"  
GraphicsWindow.FillRectangle(60, 100, 300, 60)
```

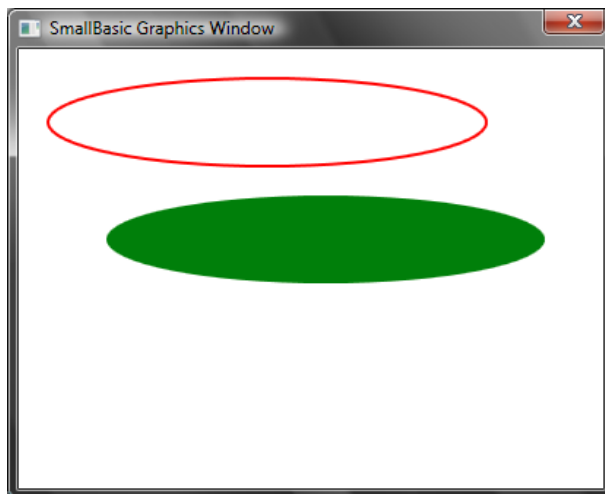


Фигура 30: Изчертаване и запълване

За да нарисувате или запълвате правоъгълник са ви необходими четири числа. Първите две представляват координатите X и Y за горния ляв ъгъл на фигурата. Третото число е ширината, докато четвъртото – височината на право-

ъгълника. Това се прилага и при чертане и запълване на елипси, както е показано в следващата програма.

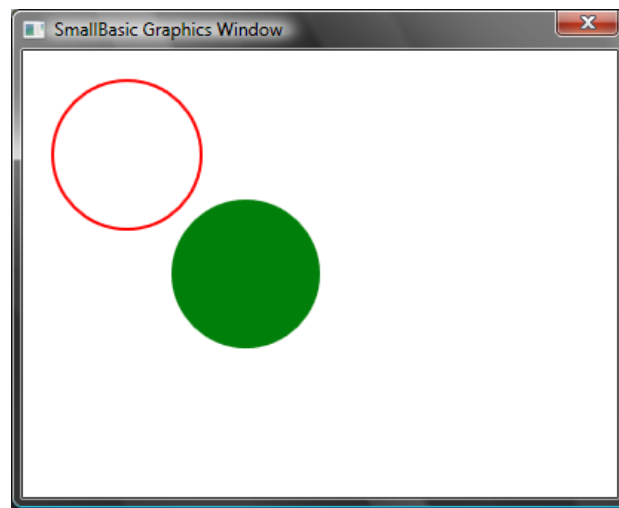
```
GraphicsWindow.Width = 400  
GraphicsWindow.Height = 300  
GraphicsWindow.PenColor = "Red"  
GraphicsWindow.DrawEllipse(20, 20, 300, 60)  
GraphicsWindow.BrushColor = "Green"  
GraphicsWindow.FillEllipse(60, 100, 300, 60)
```



Фигура 31: Изчертаване и запълване на елипси

Елипсите са общ случай на кръгове. За да начертаете кръг трябва да определите една и съща ширина и височина.

```
GraphicsWindow.Width = 400  
GraphicsWindow.Height = 300  
GraphicsWindow.PenColor = "Red"  
GraphicsWindow.DrawEllipse(20, 20, 100, 100)  
GraphicsWindow.BrushColor = "Green"  
GraphicsWindow.FillEllipse(100, 100, 100, 100)
```



Фигура 32: Кръгове

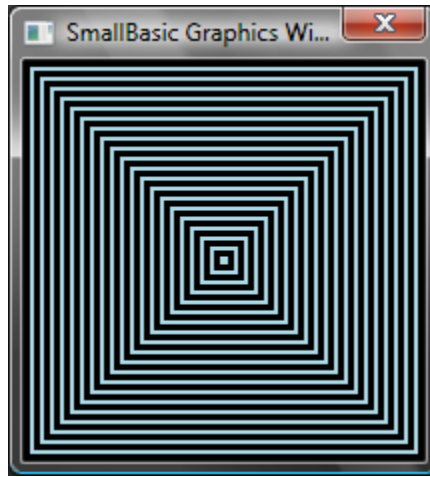
## Забавление с фигури

В тази глава ще се позабавляваме с всичко, което научихме до сега. Главата съдържа интересни примери, които комбинират всичко научено за създаване на няколко интересни програми.

### Изобилие от правоъгълници

Тук чертаем няколко правоъгълници в цикъл, с увеличаващ се размер.

```
GraphicsWindow.BackgroundColor = "Black"  
GraphicsWindow.PenColor = "LightBlue"  
GraphicsWindow.Width = 200  
GraphicsWindow.Height = 200  
For i = 1 To 100 Step 5  
    GraphicsWindow.DrawRectangle(100 - i, 100 - i, i * 2, i * 2)  
EndFor
```



Фигура 33: Изобилие от правоъгълници

## Изобилие от кръгове

Вариант на предишната програма, но с кръгове вместо квадрати.

```
GraphicsWindow.BackgroundColor = "Black"  
GraphicsWindow.PenColor = "LightGreen"  
GraphicsWindow.Width = 200  
GraphicsWindow.Height = 200  
For i = 1 To 100 Step 5  
    GraphicsWindow.DrawEllipse(100 - i, 100 - i * 2, i * 2)  
EndFor
```



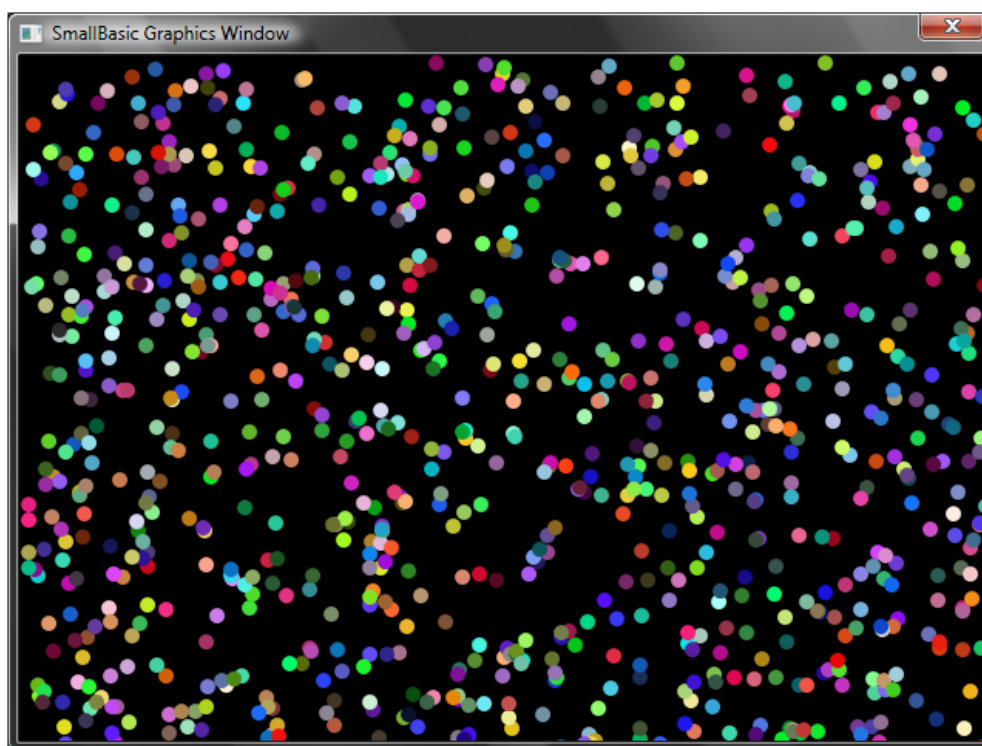
Фигура 34: Изобилие от кръгове

## Случайни цветове

Тази програма използва операцията `GraphicsWindow.GetRandomColor` за да определи произволни цветове за четката, след което - `Math.GetRandomNumber` за да определи координатите на кръговете. Тези две операции могат да бъдат

комбинирани по интересен начин и да се напишат интересни програми, които дават различни резултати при всяко тяхно стартиране.

```
GraphicsWindow.BackgroundColor = "Black"  
For i = 1 To 1000  
    GraphicsWindow.BrushColor = GraphicsWindow.GetRandomColor()  
    x = Math.GetRandomNumber(640)  
    y = Math.GetRandomNumber(480)  
    GraphicsWindow.FillEllipse(x, y, 10, 10)  
EndFor
```



Фигура 35: Случайни цветове

## Фрактали

Следващата програма чертае прост триъгълен фрактал с използване на случайни числа. Фрактал е геометрична форма, която може да бъде разделена на части, всяка от които е напълно еднаква с родителската. В този случай, програмата рисува стотици триъгълници еднакви с основния триъгълник. При изпълнение на програмата дори и за няколко секунди може да видите бавното формиране на триъгълниците само от точки. Логиката е трудна за описание за това остава упражнение за вас да я изследвате.

```
GraphicsWindow.BackgroundColor = "Black"
```

```

x = 100
y = 100

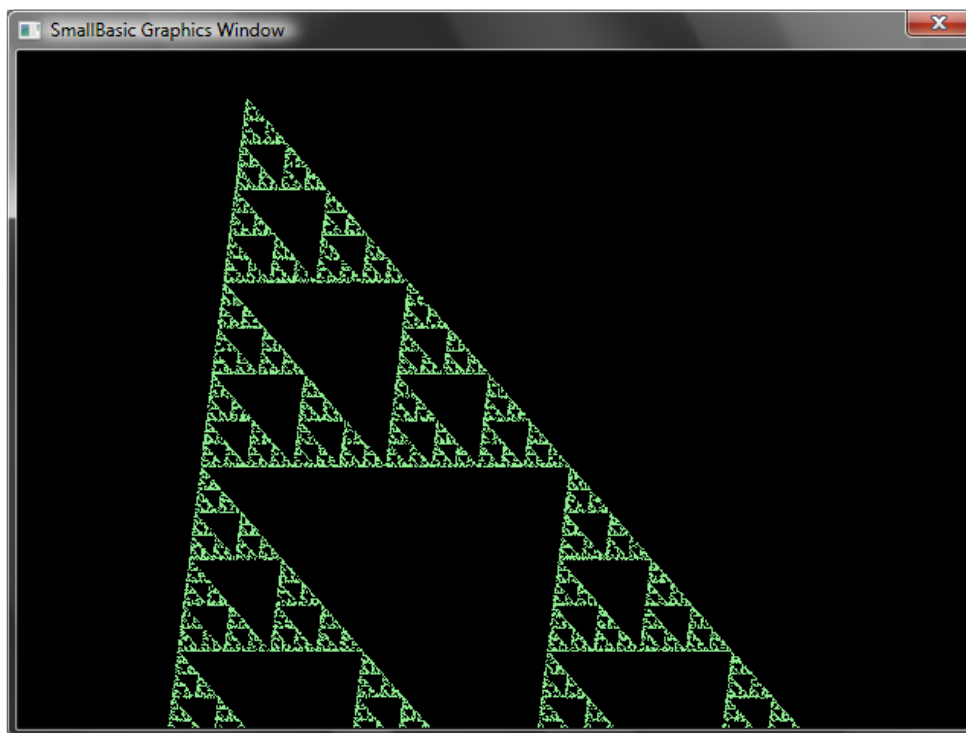
For i = 1 To 100000
    r = Math.GetRandomNumber(3)
    ux = 150
    uy = 30
    If (r = 1) then
        ux = 30
        uy = 1000
    EndIf

    If (r = 2) Then
        ux = 1000
        uy = 1000
    EndIf

    x = (x + ux) / 2
    y = (y + uy) / 2

    GraphicsWindow.SetPixel(x, y, "LightGreen")
EndFor

```



Фигура 36: Триъгълни фрактали

Ако искате наистина да видите как точките бавно оформят фрактала, може да зададете забавяне в цикъла с използване на операцията Program.Delay. Тя използва число, определено в милисекунди, представляващо времето на забавяне. По-долу е представена променена програма.

```
GraphicsWindow.BackgroundColor = "Black"
x = 100
y = 100

For i = 1 To 100000
    r = Math.GetRandomNumber(3)
    ux = 150
    uy = 30
    If (r = 1) then
        ux = 30
        uy = 1000
    EndIf

    If (r = 2) Then
        ux = 1000
        uy = 1000
    EndIf

    x = (x + ux) / 2
    y = (y + uy) / 2

    GraphicsWindow.SetPixel(x, y, "LightGreen")
    Program.Delay(2)
EndFor
```

Повишаване на забавянето прави програмата по-бавна. Експериментирайте със стойностите за да видите, коя ще бъде по Ваш вкус. Друга промяна, която може да направите е да заместите следния ред:

```
GraphicsWindow.SetPixel(x, y, "LightGreen")
```

с

```
color = GraphicsWindow.GetRandomColor()
GraphicsWindow.SetPixel(x, y, color)
```

Тази промяна ще накара програмата да рисува пикселите на триъгълниците със случайни цветове.

## Графика с костенурка

---

### Logo

През 70-те години на миналия век имаше малък, но мощен програмен език наречен Logo. Той беше използван от шепа разработчици. Това беше така докато някои не добави към езика така наречената „костенурка“. Тази костенурка беше видима на екран и отговаряше на команди като: Движи се напред, Завий наляво, Завий надясно и т.н. Използвайки костенурката хората можеха да рисуват интересни неща на екрана. Това направи езика достъпен и привлекателен за хора от различна възраст, и беше основателно широко популярен през 80-те.

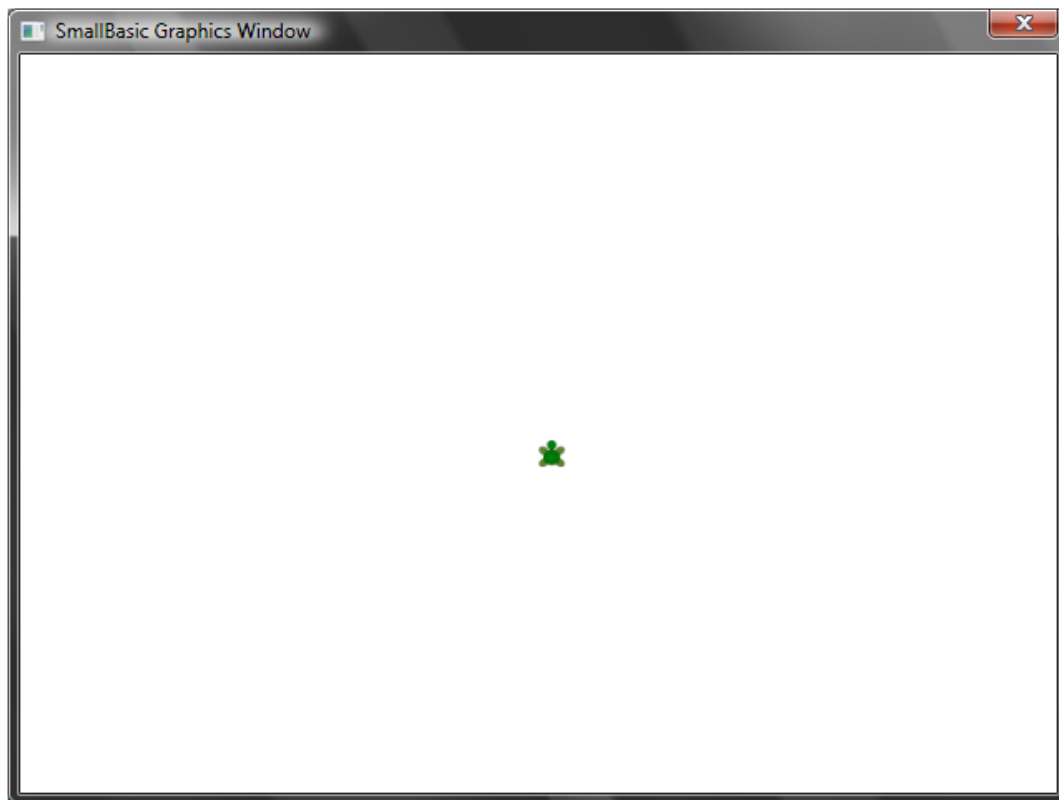
Small Basic идва с обект Turtle и много команди свързани с него, които могат да бъдат извикани в програмите на Small Basic. В тази глава ще използваме Turtle за да рисуваме графика на екрана.

### Костенурката

За да започнем трябва да изведем костенурката на екран. Това се постига с проста програма от един ред.

## Turtle.Show()

Стартирайки програмата ще забележите бял прозорец, същият като този в предишната глава, но с костенурка в центъра. Това е костенурката, която ще следва нашите инструкции и ще чертае това, което поискаме от нея.



Фигура 37: Костенурката е видима

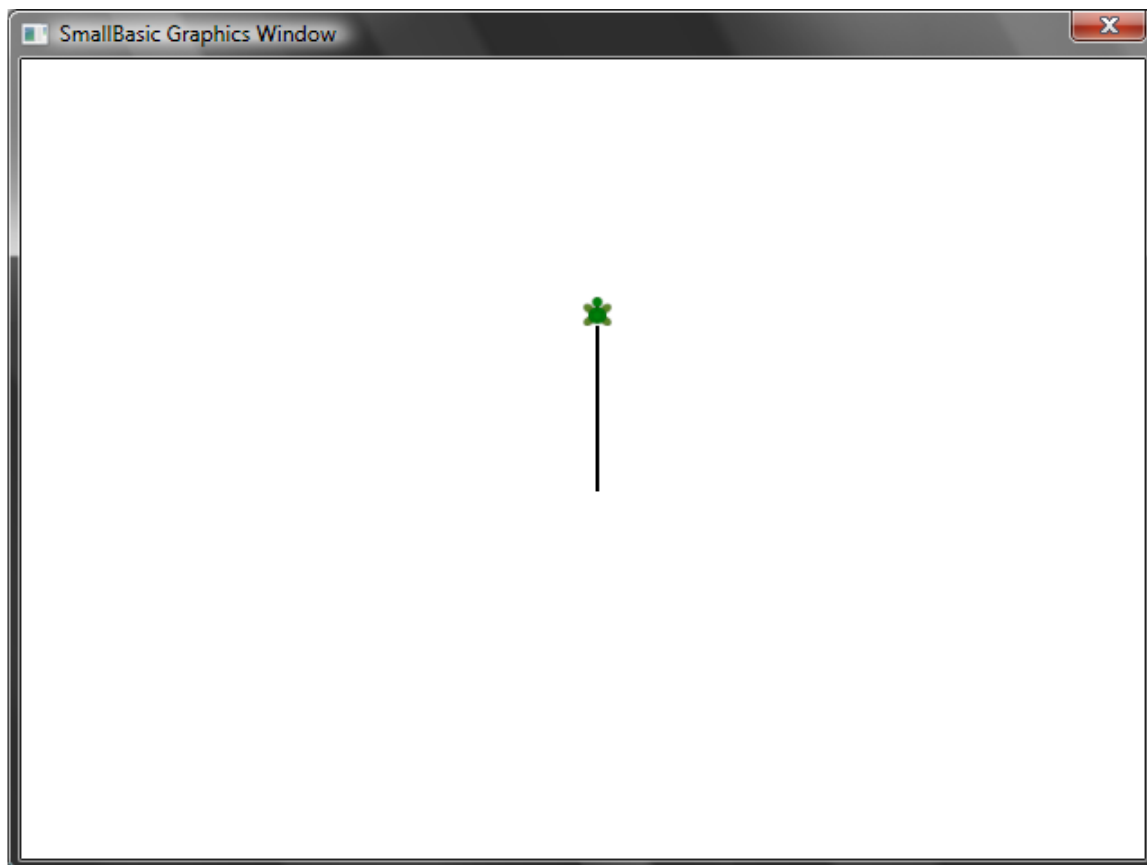
## Местене и чертане

Една от инструкциите, които костенурката разбира е **Move**. Тази операция като вход има число. То и казва на какво разстояние да се движи. В примера по-долу казваме на костенурката да се премести на разстояние 100 пиксела.

## Turtle.Move(100)

Стартирайки програмата Вие в действителност я виждате да се премества 100 пиксела напред. Движейки се тя чертае зад себе си права линия. След като спре движението резултатът е подобен на тази на следващата фигура.

*Когато се използват операции върху костенурката не е необходимо да се вика команда Show(). Тя става видима автоматично веднага след използването на която и да е операция върху нея.*



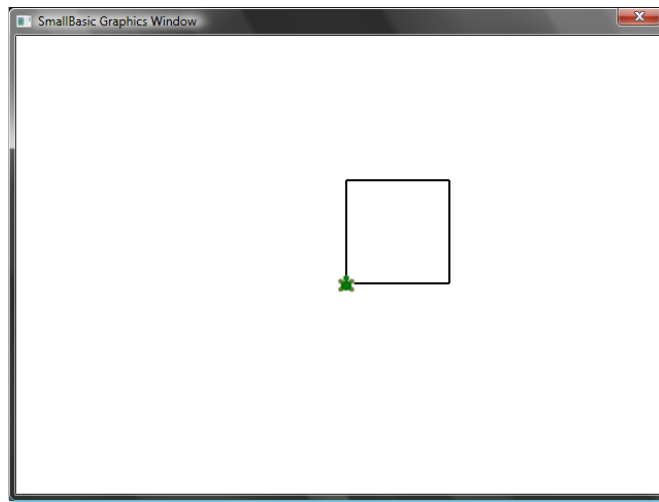
Фигура 38: Преместване на разстояние 100 пиксела

## Очертаване на квадрат

Квадратът има четири страни – две вертикални и две хоризонтални. За да постигнем изчертаването му, трябва да можем да накараме костенурката да начертае права линия, да завие на дясно и да продължи по същия начин докато и четирите страни за начертани. Ако преведем това на програмата, то би изглеждало по следния начин:

```
Turtle.Move(100)
Turtle.TurnRight()
Turtle.Move(100)
Turtle.TurnRight()
Turtle.Move(100)
Turtle.TurnRight()
Turtle.Move(100)
Turtle.TurnRight()
```

След стартирането на програмата, може да видите как костенурката чертае квадрат страна по страна и резултатът ще бъде следния:



Фигура 39: Костенурката рисува квадрат

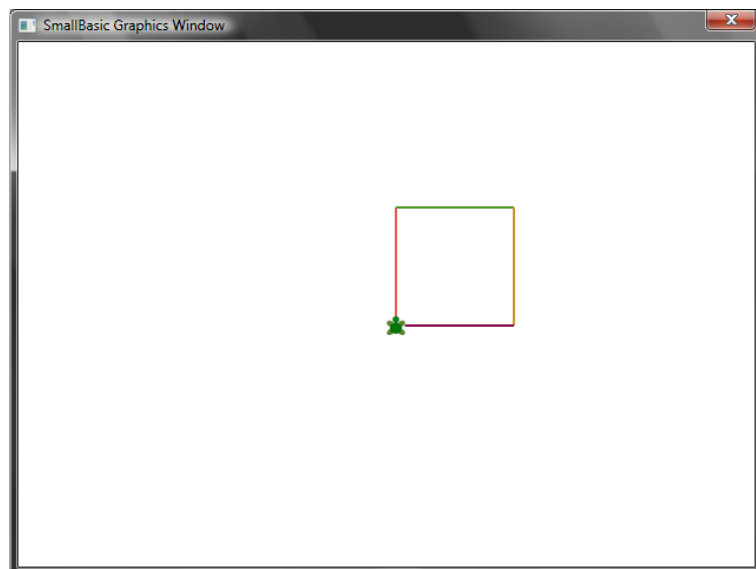
Интересно е да се отбележи, че използваме две еднакви инструкции последователно, точно четири пъти. Вече знаете, че повтарящи се команди могат да се изпълнят с използването на цикли. Така че ако преработим горната програма с цикъл **For...EndFor** ще постигнем много по-проста програма.

```
For i = 1 To 4
    Turtle.Move(100)
    Turtle.TurnRight()
EndFor
```

## Промяна на цветовете

Костенурката чертае в същия прозорец, който видяхме в предишната глава. Това означава, че всички операции, които научихме вече са валидни и тук. Например следната програма ще начертае квадрат, чиито страни са с различен цвят.

```
For i = 1 To 4
    GraphicsWindow.PenColor = GraphicsWindow.GetRandomColor()
    Turtle.Move(100)
    Turtle.TurnRight()
EndFor
```



Фигура 40: Промяна на цветовете

## Чертаене на по-сложни фигури

Като допълнение към **TurnRight** и **TurnLeft**, костенурката има и операция **Turn**. Операцията има един вход, който определя ъгъла на въртене. С нея е възможно да се начертаят полигони с различен брой страни. Следващата програма чертае хексагон (полигон със шест страни).

```
For i = 1 To 6
  Turtle.Move(100)
  Turtle.Turn(60)
EndFor
```

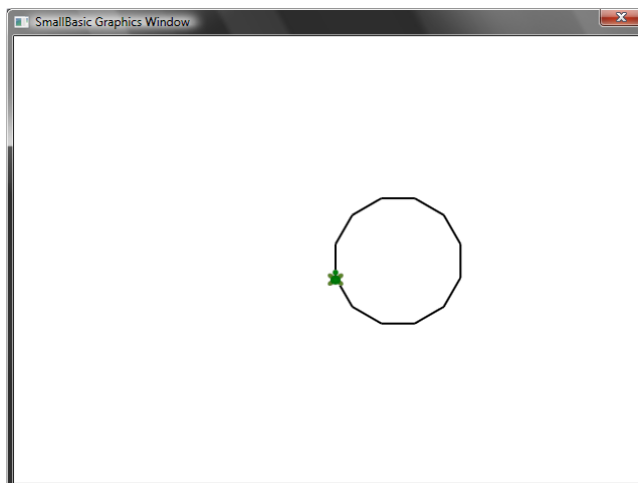
Пробвайте програмата, за да се убедите, че наистина се чертае хексагон. Забележете, че при ъгъл между страните  $60^{\circ}$  използваме **Turn(60)**. При полигон с еднаква дължина на страните, ъгълът между тях може да бъде лесно получен като се раздели 360 на броя им. Въоръжени с тази информация и използвайки променливи може да напишем обща програма чертаеща полигони с различен брой страни.

```
sides = 12

length = 400 / sides
angle = 360 / sides
```

```
For i = 1 To sides
  Turtle.Move(length)
  Turtle.Turn(angle)
EndFor
```

С използване на тази програма може да чертаете какъвто и да е полигон, като само променяте променливата **sides**. Приемайки стойност 4 получаваме квадрат. Ако стойността е достатъчно голяма, да речем 50, резултатът ще наподобява кръг.



Фигура 41: Чертаене на полигон с 12 страни

Използвайки тази техника може да начертаяме няколко кръга, с малки изменения в тях, получавайки интересен резултат.

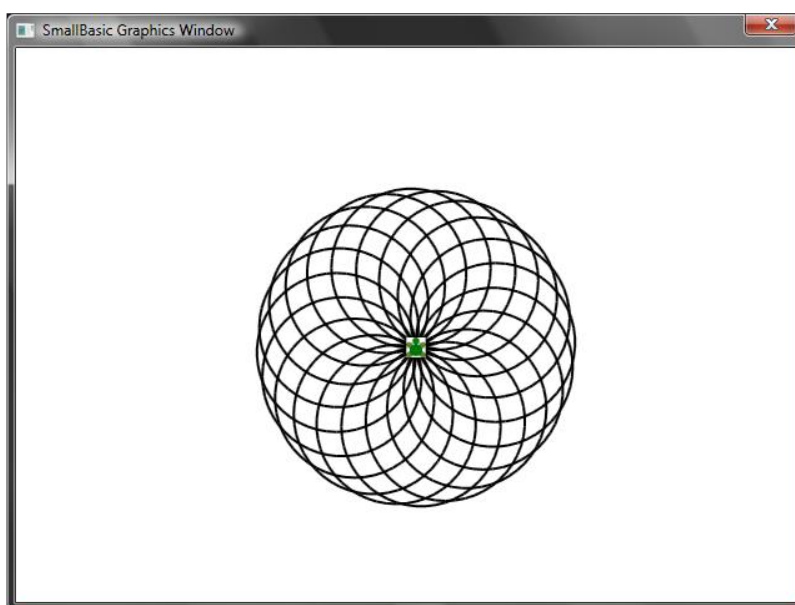
```
sides = 50
length = 400 / sides
angle = 360 / sides

Turtle.Speed = 9

For j = 1 To 20
  For i = 1 To sides
    Turtle.Move(length)
    Turtle.Turn(angle)
  EndFor
  Turtle.Turn(18)
EndFor
```

Този програма използва два цикъла **For..EndFor** един в друг. Вътрешният цикъл ( $i = 1 \text{ to } sides$ ) е подобен на този от програмата с полигона и отговаря за чертаенето на кръга. Външният цикъл ( $j = 1 \text{ to } 20$ ) обръща костенурката за всеки начертан кръг. Заради него тя чертае 20 кръга. Поставени заедно в една програма имат за резултат интересна шарка, като тази показана на фигурата.

*В горната програма костенурката се движи по-бързо, защото скоростта (*Speed*) е променена на 9. Можете да промените това свойство с всяка стойност между 1 и 10 и да я накарате да се движи толкова бързо колкото желаете.*



Фигура 42: Фигура в кръгове

## Движение в кръг

За да не рисува костенурката се използва **PenUp**. Това ви позволява да я преместите където пожелаете на екрана без да чертаете линии. **PenDown** ще я накара да рисува отново. Може да се използва при постигане на някакви специални ефекти, като например прекъснати линии. Пример за това е следващата програма, с която се чертае полигон с прекъснатата линия.

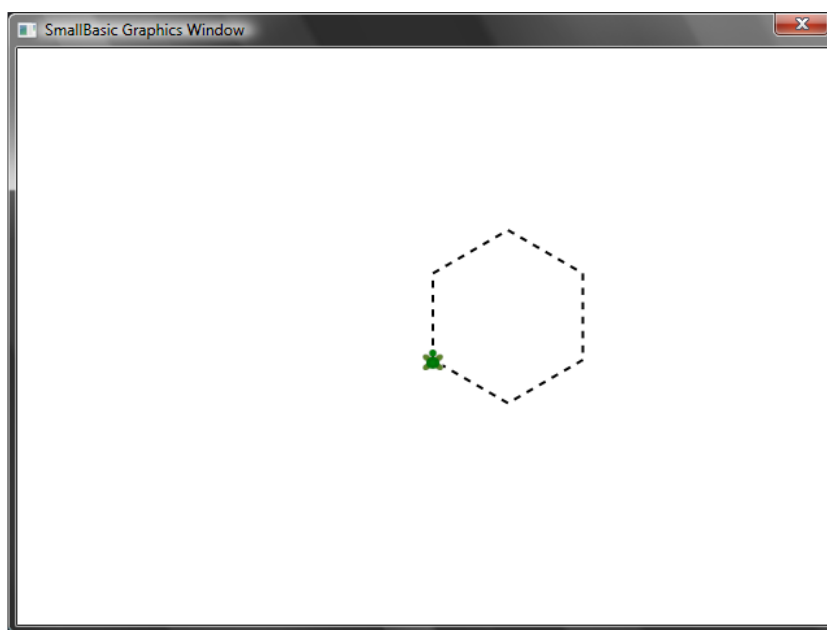
```
sides = 6
```

```
length = 400 / sides
```

```
angle = 360 / sides
```

```
For i = 1 To sides
  For j = 1 To 6
    Turtle.Move(length / 12)
    Turtle.PenUp()
    Turtle.Move(length / 12)
    Turtle.PenDown()
  EndFor
  Turtle.Turn(angle)
EndFor
```

Отново имаме два цикъла. Вътрешният чертае една прекъсната линия, докато. Външният определя колко линии да се начертаят. В примера за променливата **sides** използваме числото 6, следователно имаме Хексагон като този показан на фигурата.



Фигура 43: Използване на PenUp и PenDown

## Глава 9

# Под-процедури

---

Пишейки програми често ще попадаме на случаи, в които трябва да се изпълни една и съща група от стъпки последователно няколко пъти. Тогава вероятно не би имало смисъл да се пишат едни и същи изрази няколко пъти. В такива случаи под-процедурите са под ръка.

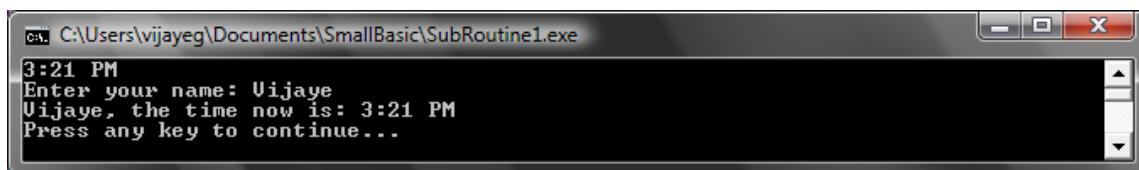
Под-процедурата е част от код в по-голяма програма, която обикновено прави нещо много специфично и може да бъде извикана от където и да е в програмата. Под-процедурите се определят от име, което следва ключовата дума **Sub** и завършват с ключовата дума **EndSub**. Например следният откъс представя под-процедура, с името *PrintTime*. Тя печата текущото време в текстовия прозорец.

```
Sub PrintTime
    TextWindow.WriteLine(Clock.Time)
EndSub
```

По долу е показана програма използваща процедурата и извикваща я от различни места.

```
PrintTime()
TextWindow.Write("Enter your name: ")
name = TextWindow.Read()
TextWindow.Write(name + ", the time now is: ")
PrintTime()
```

```
Sub PrintTime
    TextWindow.WriteLine(Clock.Time)
EndSub
```



Фигура 44: Извикване на проста под-процедура

Под-процедура се изпълнява със *SubroutineName()*. Както обикновено пунктуационните знаци „()“ са необходими за да може да разбере компютъра че искате да изпълните под-процедура.

## Предимства от използването на под-процедури

Както току що видяхме под-процедурите помагат за намаляването на кода, който трябва да напишете. Когато под-процедурата *PrintTime*, можете да я извикате от където и да е в програмата и тя ще напечата текущото време.

*Запомнете! Можете да извикате под-процедура в Small Basic само в рамките на една и съща програма. Това не може да стане между различни програми.*

В допълнение под-променливите могат да помогнат за разделянето на сложни проблеми на по-малки части. Да предположим, че имате да разрешите сложно уравнение. Можете да напишете под-процедури, които решават малки части от него. След което събирате резултатите и получавате решението на оригиналното уравнение.

Под-процедурите спомагат и да се подобри яснотата на програмите. С други думи ако имате добре наименувани под-процедури за често изпълнявани части от Вашата програма, то тя става ясна и разбираема. Това е особено важно ако желаете да разберете нечия чужда програма или желаете Вашата програма да е разбираема за други. Дори е полезно при случаи, когато желаете да прочетете собствената си програма седмици след като сте я написали.

## Използване на променливи

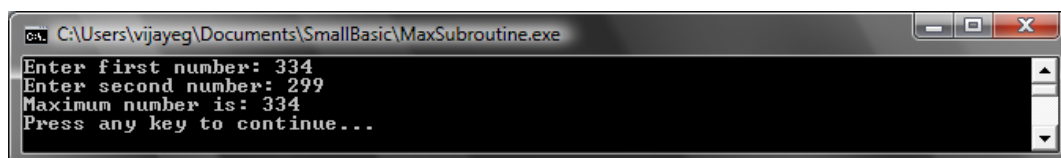
В под-процедурата имате достъп до всяка променлива, използвана в програмата ви. Като пример следната програма приема две числа и извежда по-голямото от тях. Забележете, че променливата *max* е използвана както вътре така и извън под-процедурата.

```
TextWindow.Write("Enter first number: ")
num1 = TextWindow.ReadNumber()
TextWindow.Write("Enter second number: ")
num2 = TextWindow.ReadNumber()

FindMax()
TextWindow.WriteLine("Maximum number is: " + max)

Sub FindMax
    If (num1 > num2) Then
        max = num1
    Else
        max = num2
    EndIf
EndSub
```

И изхода на програмата изглежда така.



Фигура 45: извеждане на по-голямото от две числа с използване на под-процедура

Нека разгледаме още един пример, който илюстрира използването на под-процедури. Този път ще разгледаме графична програма, изчисляваща различни точки и ги съхранява в променливите *x* и *y*. След това се извиква под-процедурата **DrawCircleUsingCenter**, използваща *x* и *y* за център на начертания от нея кръг.

```
GraphicsWindow.BackgroundColor = "Black"
GraphicsWindow.PenColor = "LightBlue"
GraphicsWindow.Width = 480
For i = 0 To 6.4 Step 0.17
```

```

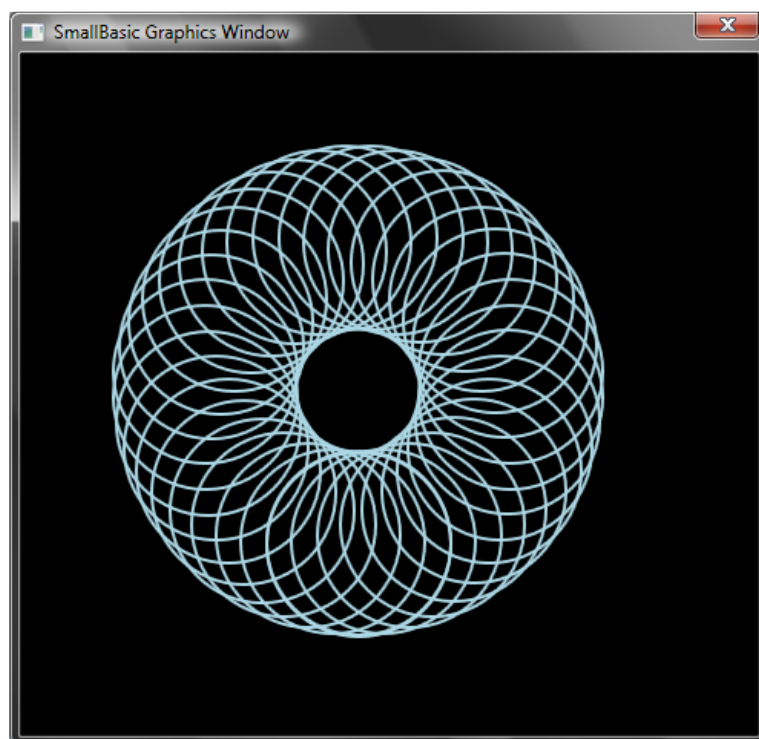
x = Math.Sin(i) * 100 + 200
y = Math.Cos(i) * 100 + 200

DrawCircleUsingCenter()
EndFor

Sub DrawCircleUsingCenter
    startX = x - 40
    startY = y - 40

    GraphicsWindow.DrawEllipse(startX, startY, 120, 120)
EndSub

```



Фигура 46: Графичен пример за под-процедура

## Извикване на под-процедури в цикли

Понякога под-процедурите се извикват в цикъл. През това време те изпълняват една и съща група от изрази, но с различни стойности в една или няколко променливи. Например нека имаме под-процедура с име PrimeCheck и тази под-процедура определя дали едно число е просто или не. Използвайки тази под-процедура може да напишете програма, която позволява на потребителя да въведе стойност и чрез нея да се определи дали тя е проста или не.

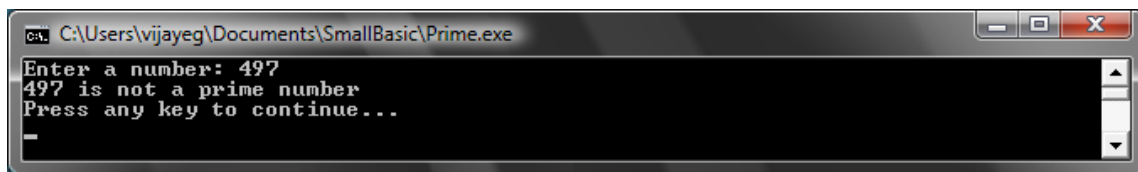
```

TextWindow.Write("Enter a number: ")
i = TextWindow.ReadNumber()
isPrime = "True"
PrimeCheck()
If (isPrime = "True") Then
    TextWindow.WriteLine(i + " is a prime number")
Else
    TextWindow.WriteLine(i + " is not a prime number")
EndIf

Sub PrimeCheck
    For j = 2 To Math.SquareRoot(i)
        If (Math.Remainder(i, j) = 0) Then
            isPrime = "False"
            Goto EndLoop
        EndIf
    Endfor
EndLoop:
EndSub

```

Под-процедурата PrimeCheck взима стойността  $i$  и се опитва да я раздели на по-малки числа. Ако число раздели  $i$  и няма остатък, то  $i$  не е просто число. В този момент под-процедурата поставя стойност за *isPrime* за „False“ и излиза. Ако числото е неделимо от по-малки числа то стойността остава „True“.



Фигура 47: Проверка на прости числа

Сега вече имате под-процедура, която проверява дали едно число е просто. Желаете обаче да използвате това в програма, която извежда списък с всички прости числа например под 100. Лесно е да преработите програмата по-горе и да извикате под-процедурата в цикъл. Така на нея се дава различна стойност за пресмятане всеки път, когато цикълът се изпълни. Да видим как е решено това в примера по-долу.

```

For i = 3 To 100
isPrime = "True"

```

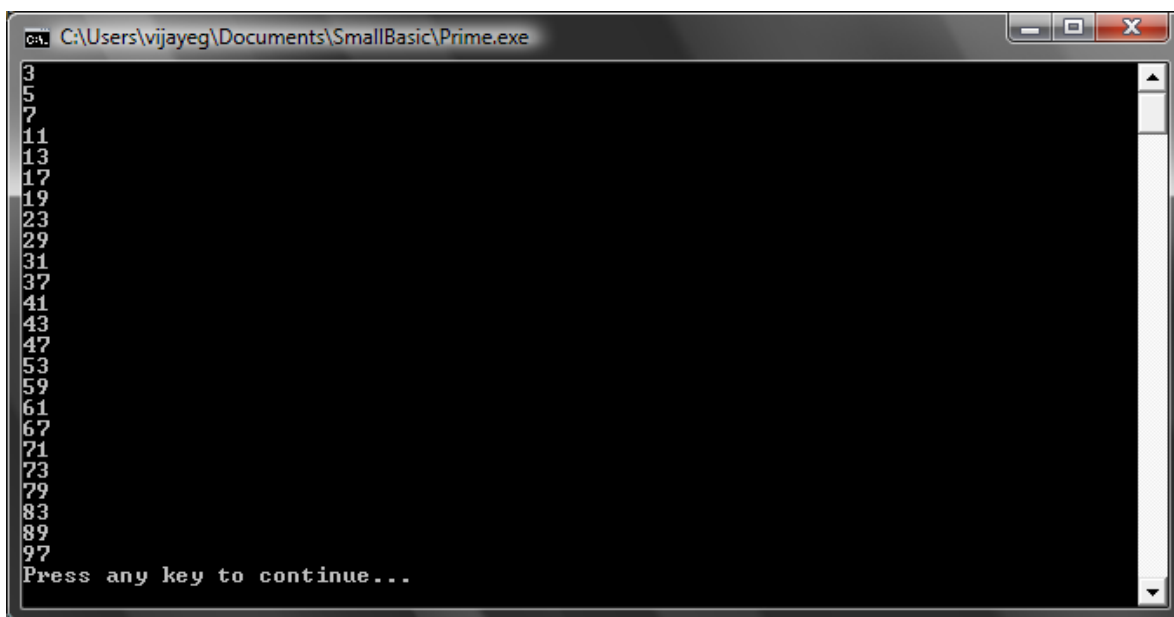
```

PrimeCheck()
If (isPrime = "True") Then
    TextWindow.WriteLine(i)
EndIf
EndFor

Sub PrimeCheck
    For j = 2 To Math.SquareRoot(i)
        If (Math.Remainder(i, j) = 0) Then
            isPrime = "False"
            Goto EndLoop
        EndIf
    Endfor
EndLoop:
EndSub

```

Тук стойността на  $i$  е обновяване с всяко изпълнение на цикъла. В цикъла е извикана под-процедурата *PrimeCheck*. Тя взима стойността на  $i$  и изчислява дали  $i$  е просто число. Резултатът се запазва в променливата *isPrime*, която вече е достъпна за цикъла извън под-процедурата. Ако се окаже, че  $i$  е просто число, стойността ѝ се извежда на екран. И понеже цикъла започва от 3 и продължава до 100, имаме списък с всички прости числа от 3 до 100. Резултатът е показан на фигурата:



Фигура 48: Прости числа

## Глава 10

# Масиви

---

До сега Вие трябва да сте научили как да използвате променливи и в края на краищата, сте достигнали до тук. Забавлявате се, нали?

Нека си припомним първата програма, която написахме с променливи.

```
TextWindow.Write("Enter your Name: ")
name = TextWindow.Read()
TextWindow.WriteLine("Hello " + name)
```

В тази програма ние получавахме и съхранявахме името на потребителя в променлива с името **name**. По-късно казахме „Здравей“ на потребителя. Сега нека предположим, че има повече от един потребител – да речем те са 5. Как да съхраним техните имена? Единият начин е:

```
TextWindow.Write("User1, enter name: ")
name1 = TextWindow.Read()
TextWindow.Write("User2, enter name: ")
name2 = TextWindow.Read()
TextWindow.Write("User3, enter name: ")
name3 = TextWindow.Read()
TextWindow.Write("User4, enter name: ")
name4 = TextWindow.Read()
TextWindow.Write("User5, enter name: ")
name5 = TextWindow.Read()
TextWindow.WriteLine("Hello ")
```

```
TextWindow.Write(name1 + ", ")
TextWindow.Write(name2 + ", ")
TextWindow.Write(name3 + ", ")
TextWindow.Write(name4 + ", ")
TextWindow.WriteLine(name5)
```

След стартиране на програмата получавате следния резултат:



```
C:\Users\vijayeg\AppData\Local\Temp\tmp25DA.tmp.exe
User1, enter name: Shifu
User2, enter name: Tigress
User3, enter name: Po
User4, enter name: Oogway
User5, enter name: Mantis
Hello Shifu, Tigress, Po, Oogway, Mantis
Press any key to continue...
```

Фигура 49: Без използване на масиви

Очевидно има по-лесен начин да се напише тази програма. Особено след като компютърът е много добър в изпълнението на повтарящи се задачи, защо да се притесняваме да пишем един и същ код за всеки един от потребителите? Трикът тук е да съхраняваме и възвръщаме повече от едно име използвайки една променлива. Ако можем това ще използваме научения преди цикъл **For**. Това е момента, когато масивите идват на помощ.

## Какво е масив?

Масив е специфична променлива, която в определен момент може да съхранява повече от една стойност. Това означава, че вместо да използваме променливите **name1**, **name2**, **name3**, **name4** и **name5**, за да съхраняваме 5 имена, може да използваме само **name** за да съхраним всички. Начинът за съхраняване на няколко стойности се състои в използването на „индекс“. Например всяка една от **name[1]**, **name[2]**, **name[3]**, **name[4]** и **name[5]** може да съхранява променлива. Номерата 1, 2, 3, 4 и 5 се наричат индекси на масива.

Въпреки че **name[1]**, **name[2]**, **name[3]**, **name[4]** и **name[5]** изглеждат като различни променливи, те всъщност са една. Може да попитате какво е предимството на това? Най-хубавото да се съхраняват стойности в масиви е че

индекси могат да се определят с друга променлива, което ни дава възможност да имаме достъп до масива в цикли.

Нека да видим как да използваме натрупаното знание и да пренапишем нашата програма с използване на променливи.

```
For i = 1 To 5
    TextWindow.Write("User" + i + ", enter name: ")
    name[i] = TextWindow.Read()
EndFor

TextWindow.Write("Hello ")
For i = 1 To 5
    TextWindow.Write(name[i] + ", ")
EndFor
TextWindow.WriteLine("")
```

Много по лесно да се прочете, нали? Забележете двата болдирани реда. Първият съхранява стойност в масива, вторият чете от него. Стойността съхранена в **name[1]** няма да бъде повлияна от тази съхранена в **name[2]**. От тук насетне в повечето случаи можете да гледате на **name[1]** и **name[2]** като на различни променливи с една и съща идентичност.



Фигура 50: С използване на масиви

Програмата дава почти същия резултат както и тази без масиви с изключение на запетайката след последното име. Това може да бъде оправено с пренаписване на печатащия цикъл по следния начин:

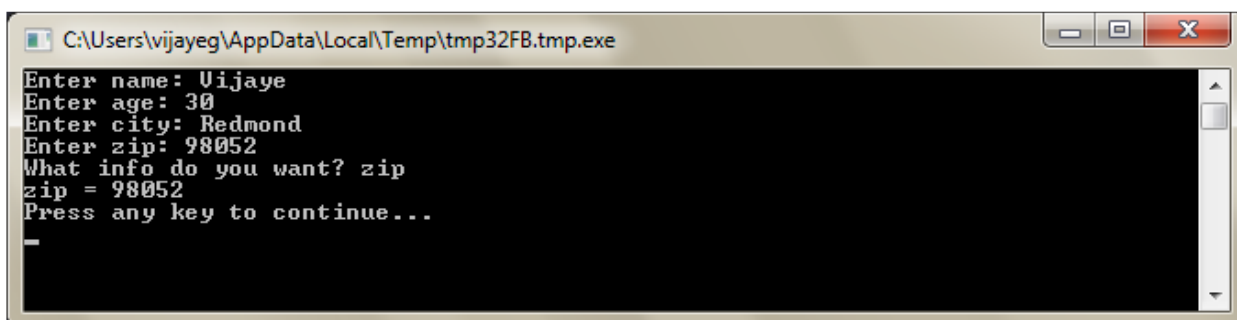
```
TextWindow.Write("Hello ")
```

```
For i = 1 To 5
    TextWindow.Write(name[i])
    If i < 5 Then
        TextWindow.Write(", ")
    EndIf
EndFor
TextWindow.WriteLine("")
```

## Индексиране на масиви

В предишната глава видяхте, че използвахме числа за индексиране, за да съхраняваме и получаваме стойност от масиви. Индексите не са ограничени само до числа. В някои случаи дори е полезно да се използва текст за индексиране. Например в следващата програма изискваме различна информация от потребителя, след което извеждаме на екран информацията, която потребителят изисква.

```
TextWindow.Write("Enter name: ")
user["name"] = TextWindow.Read()
TextWindow.Write("Enter age: ")
user["age"] = TextWindow.Read()
TextWindow.Write("Enter city: ")
user["city"] = TextWindow.Read()
TextWindow.Write("Enter zip: ")
user["zip"] = TextWindow.Read()
TextWindow.Write("What info do you want? ")
index = TextWindow.Read()
TextWindow.WriteLine(index + " = " + user[index])
```



Фигура 51: Използване на не-цифрени индекси

## Повече от едно измерение

Да предположим, че искате да съхраните имената и телефонните номера на всички Ваши приятели и да имате възможност да търсите определен номер сред тях когато е необходимо, също както в телефонен указател. Как бихме подхождали при написването на такава програма.

В този случай има две групи от индекси, наречени измерения на масива. Да приемем, че идентифицираме всеки приятел с неговия прякор. Това става в първия индекс в масива. След като вед-

*Индексите в масиви не се влияят от главни букви. Като и в обикновени променливи, индексите може и да не си съответстват в главните букви.*

нъж сме установили кой е приятеля Ви, вторият индекс – **phone** ще ни помогне да установим и търсения телефонен номер.

Начина, по който съхраняваме информацията ще изглежда така:

```
friends["Rob"]["Name"] = "Robert"
friends["Rob"]["Phone"] = "555-6789"

friends["VJ"]["Name"] = "Vijaye"
friends["VJ"]["Phone"] = "555-4567"

friends["Ash"]["Name"] = "Ashley"
friends["Ash"]["Phone"] = "555-2345"
```

Масив, в който има две групи от индекси, се нарича масив с две измерения.

След като сме конструирали програмата, може да вземем като вход псевдонима на приятел, след което да изведем на екран информацията, която е съхранена за него. Ето и цялата програма:

```
friends["Rob"]["Name"] = "Robert"
friends["Rob"]["Phone"] = "555-6789"

friends["VJ"]["Name"] = "Vijaye"
friends["VJ"]["Phone"] = "555-4567"
```

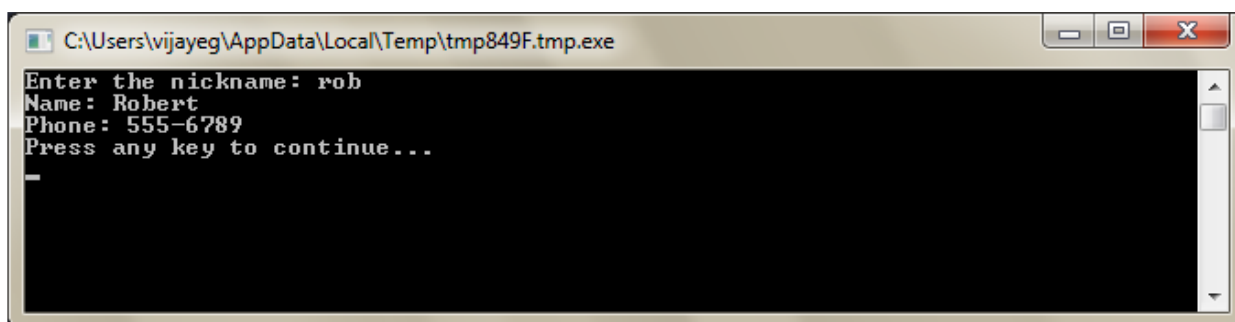
```

friends["Ash"]["Name"] = "Ashley"
friends["Ash"]["Phone"] = "555-2345"

TextWindow.Write("Enter the nickname: ")
nickname = TextWindow.Read()

TextWindow.WriteLine("Name: " + friends[nickname]["Name"])
TextWindow.WriteLine("Phone: " + friends[nickname]["Phone"])

```



Фигура 52: Опростен телефонен указател

## Използване на масиви за изобразяване на решетки

Обичайна употреба на масиви е в случай на изобразяване на решетки или таблици. Решетките имат редове и колони, което съответства на двуизмерен масив. Проста програма, която поставя четириъгълници в решетка е:

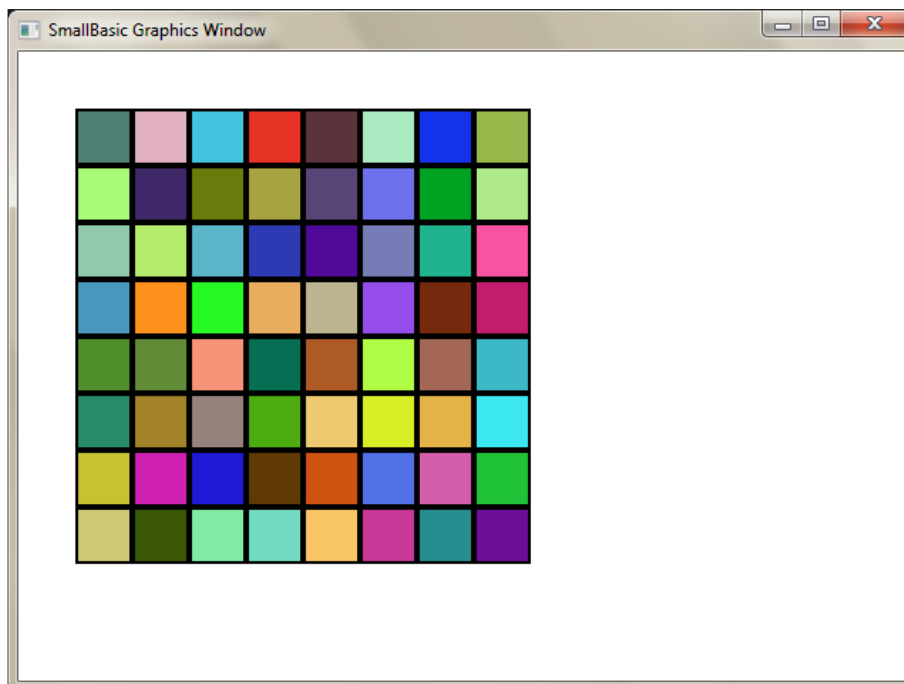
```

rows = 8
columns = 8
size = 40

For r = 1 To rows
    For c = 1 To columns
        GraphicsWindow.BrushColor = GraphicsWindow.GetRandomColor()
        boxes[r][c] = Shapes.AddRectangle(size, size)
        Shapes.Move(boxes[r][c], c * size, r * size)
    EndFor
EndFor

```

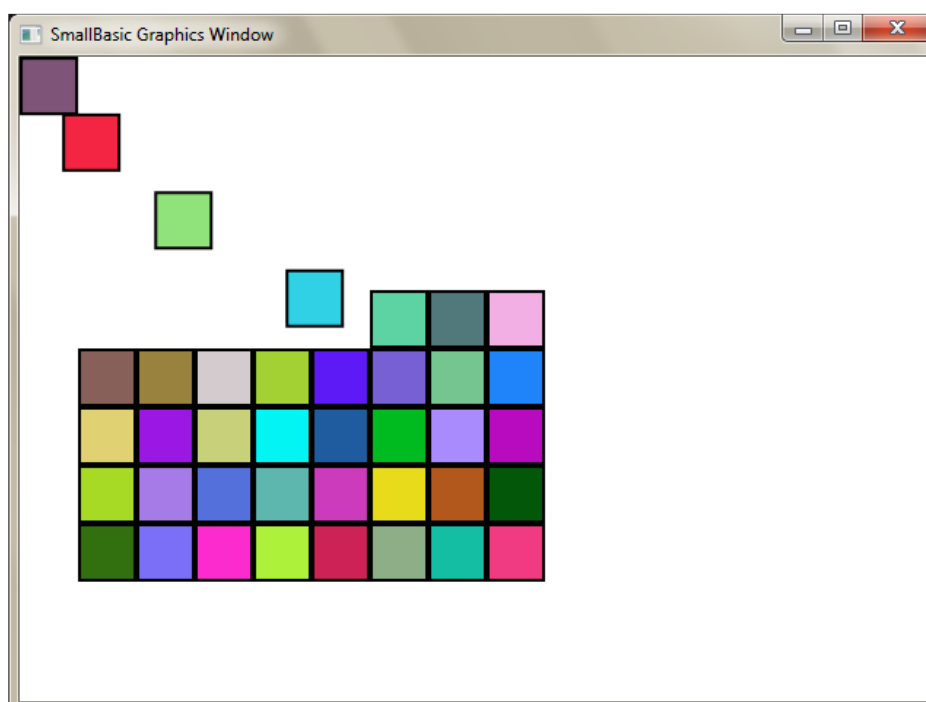
Програмата разполага правоъгълници и ги поставя в мрежа от 8 реда и 8 колони.



Фигура 53: Разполагане на правоъгълници в мрежа

Ако поставим следващия код в края на програмата, ще анимираме правоъгълниците, задвижвайки ги към горния ляв ъгъл.

```
For r = 1 To rows
  For c = 1 To columns
    Shapes.Animate[boxes[r][c], 0, 0, 1000)
    Program.Delay(300)
  EndFor
EndFor
```



Фигура 54: Проследяване пътя на правоъгълниците в мрежата

# Събития и интерактивност

---

В първите две глави представихме обекти, които имат свойства и операции. Освен тях някои обекти имат т.н. събития. Събитията са сигнали, които се появяват в отговор на действия на потребителя, например преместване или щракане с мишката. В определен смисъл събитията са противоположни на операциите. В случая на операция, Вие като програмисти я извиквате за да накарате компютъра да направи нещо. От друга страна в случая на събитие, компютърът ви известява ако нещо интересно се случва.

## В какво са полезни събитията?

Събитията са свързани с интерактивността на програмата. За да накарате потребителя да взаимодейства с програмата са ви нужни събития. Например искате да напишете игра **Tic-Tac-Toe**. Искате да накарате потребителя да избере своето ниво, нали? Тук се намесват събитията. Вие получавате това, което въвежда потребителя и от където програмата използва събития. Ако това ви се струва трудно за възприемане не се тревожете, ще разгледаме много прост пример, с който ще разберете добре какво са събитията и за какво могат да бъдат използвани. По-долу е показана елементарна програма състояща се от един израз и една под-процедура. Под-процедурата използва операцията

*ShowMessage* на обекта *GraphicsWindow* за да покаже съобщение на потребителя.

```
GraphicsWindow.MouseDown = OnMouseDown  
  
Sub OnMouseDown  
    GraphicsWindow.ShowMessage("You Clicked.", "Hello")  
EndSub
```

Интересен е редът, в който задаваме името на под-процедурата към събитието **MouseDown** на обекта *GraphicsWindow*. Забелязвате, че *MouseDown* изглежда почти като свойство, с разликата че вместо стойност и задаваме под-процедурата *OnMouseDown*. Това е и интересното в събитията – когато стане събитие се извиква под-процедура. В конкретния случай всеки път, когато потребителят щракне с мишката върху графичния прозорец се извиква под-процедурата *OnMouseDown*. Продължете, стартирайте програмата и опитайте това. Всеки път когато щракнете с мишката върху графичния прозорец виждате съобщение подобно на това:



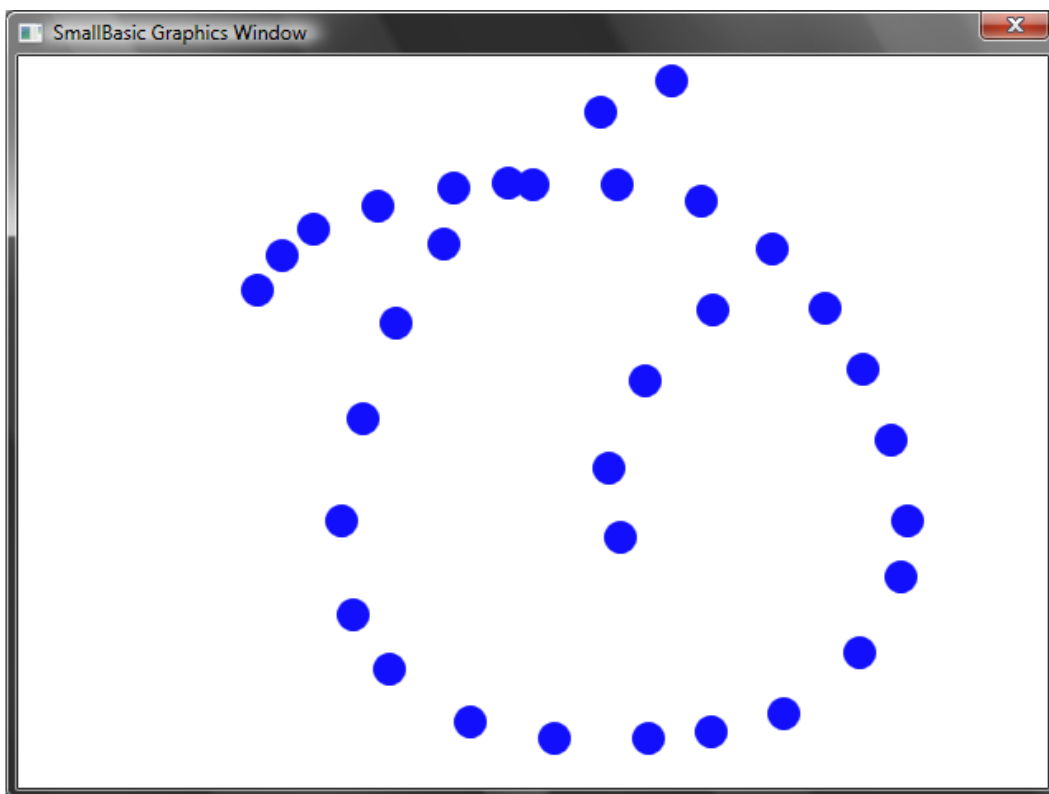
Фигура 55: Отговор на събитие

Използването на събития по този начин позволява създаването на интересни програми. Програми написани по този начин често се наричат събитийни.

Може да модифицирате под-процедурата *OnMouseDown*, така че да изпълнява други задачи освен да показва прозорец със съобщение. В следващия пример при щракане с мишката върху графичния прозорец се изчертават големи сини точки.

```
GraphicsWindow.BrushColor = "Blue"  
GraphicsWindow.MouseDown = OnMouseDown  
Sub OnMouseDown  
    x = GraphicsWindow.MouseX - 10
```

```
y = GraphicsWindow.MouseY - 10  
GraphicsWindow.FillEllipse(x, y, 20, 20)  
EndSub
```



Фигура 56: Управление на събитието `MouseDown`

Забележете, че използваме *MouseX* и *MouseY*, за да получим координатите на мишката. Използваме това за да начертаяме кръгове с център тези координати.

## Управление на няколко събития

Няма граници какъв е броя на събитията, които управлявате. Може дори една под-процедура да управлява няколко събития. Можете обаче да управлявате събитие само веднъж. Ако опитате да зададете две под-процедури на едно събитие, то втората ще спечели. За да илюстрираме това нека вземем предишния пример и да добави под-процедура, която да управлява натискането на клавиш от клавиатурата. И нека новата под-процедура да променя цвета на четката, така че при всяко щракане с мишката да се получават точки с различен цвят.

```
GraphicsWindow.BrushColor = "Blue"
```

```
GraphicsWindow.MouseDown = OnMouseDown
```

```
GraphicsWindow.KeyDown = OnKeyDown
```

```
Sub OnKeyDown
```

```
    GraphicsWindow.BrushColor = GraphicsWindow.GetRandomColor()
```

```
EndSub
```

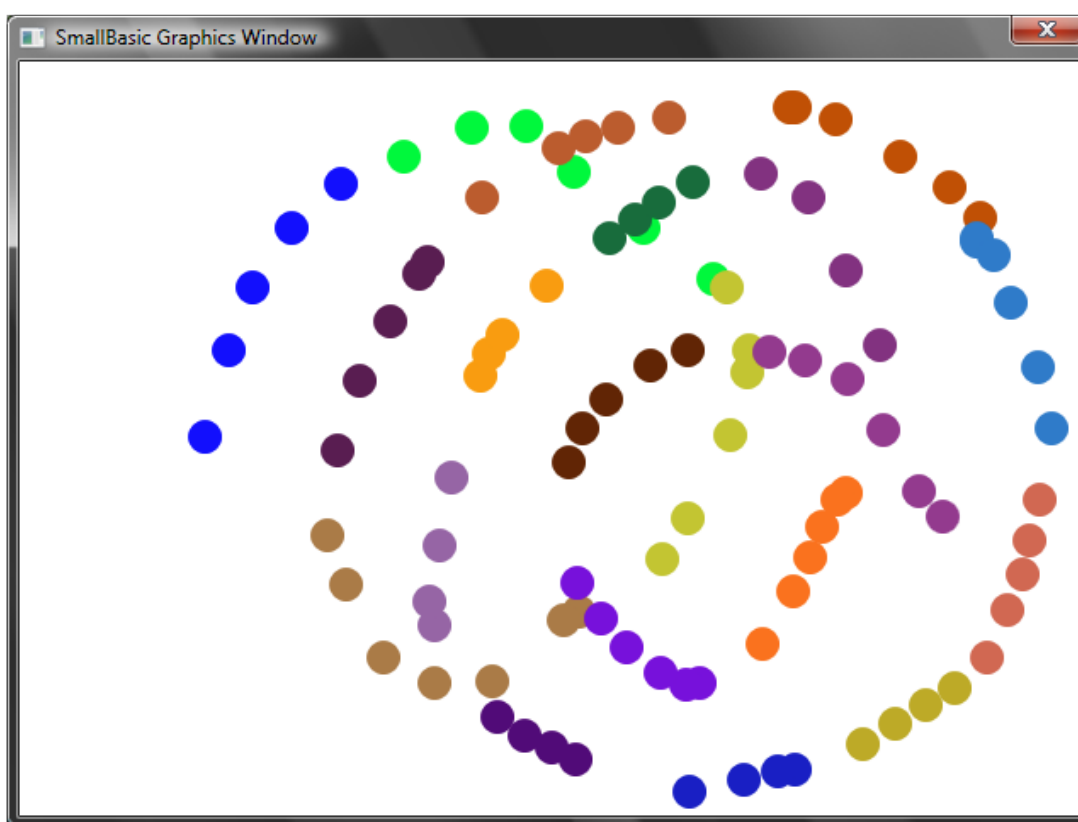
```
Sub OnMouseDown
```

```
    x = GraphicsWindow.MouseX - 10
```

```
    y = GraphicsWindow.MouseY - 10
```

```
    GraphicsWindow.FillEllipse(x, y, 20, 20)
```

```
EndSub
```



Фигура 57: Управление на няколко събития

След стартиране на програмата, ако щракнете върху прозореца веднъж ще получите синя точка. Ако натиснете произволен клавиш и щракнете отново с мишката ще имате точка с различен цвят. При натискането на клавиш се изпълнява под-процедурата *OnKeyDown*, която променя цвета на четката с произволен такъв. След щракане с мишката се изчертава кръг с новия цвят. Резултатът е група от точки с различен цвят.

## Рисувателна програма

Въоръжени със събития и под-процедури, вече може да напишем програма, която позволява на потребителя да рисува върху графичния прозорец. Изненадващо лесно е да се напише подобна програма, само ако разделим проблема на по-малки части. Като първа стъпка нека напишем програма, която позволява на потребителя да движи мишката навсякъде по графичния прозорец, оставяйки следа където е преминала.

```
GraphicsWindow.MouseMove = OnMouseMove
```

```
Sub OnMouseMove
```

```
    x = GraphicsWindow.MouseX
```

```
    y = GraphicsWindow.MouseY
```

```
    GraphicsWindow.DrawLine(prevX, prevY, x, y)
```

```
    prevX = x
```

```
    prevY = y
```

```
EndSub
```

При стартиране на програмата обаче, първата линия винаги започва от горния ляв ъгъл, координати (0, 0). За да оправим това може да използваме събитието *MouseDown* и запишем стойностите на *prevX* и *prevY* , когато то се случи.

Също така следа ни е необходима само, когато потребителят държи натиснат бутона на мишката. В другите случаи линии не трябва да се чертаят. За да получим този ефект използваме свойството *IsLeftButtonDown* на обекта **Mouse**. Това свойство определя дали левият бутон е задържан или не. Ако стойността му е “истина” то линия се рисува, ако не тя се пропуска.

```
GraphicsWindow.MouseMove = OnMouseMove
```

```
GraphicsWindow.MouseDown = OnMouseDown
```

```
Sub OnMouseDown
```

```
    prevX = GraphicsWindow.MouseX
```

```
    prevY = GraphicsWindow.MouseY
```

```
EndSub
```

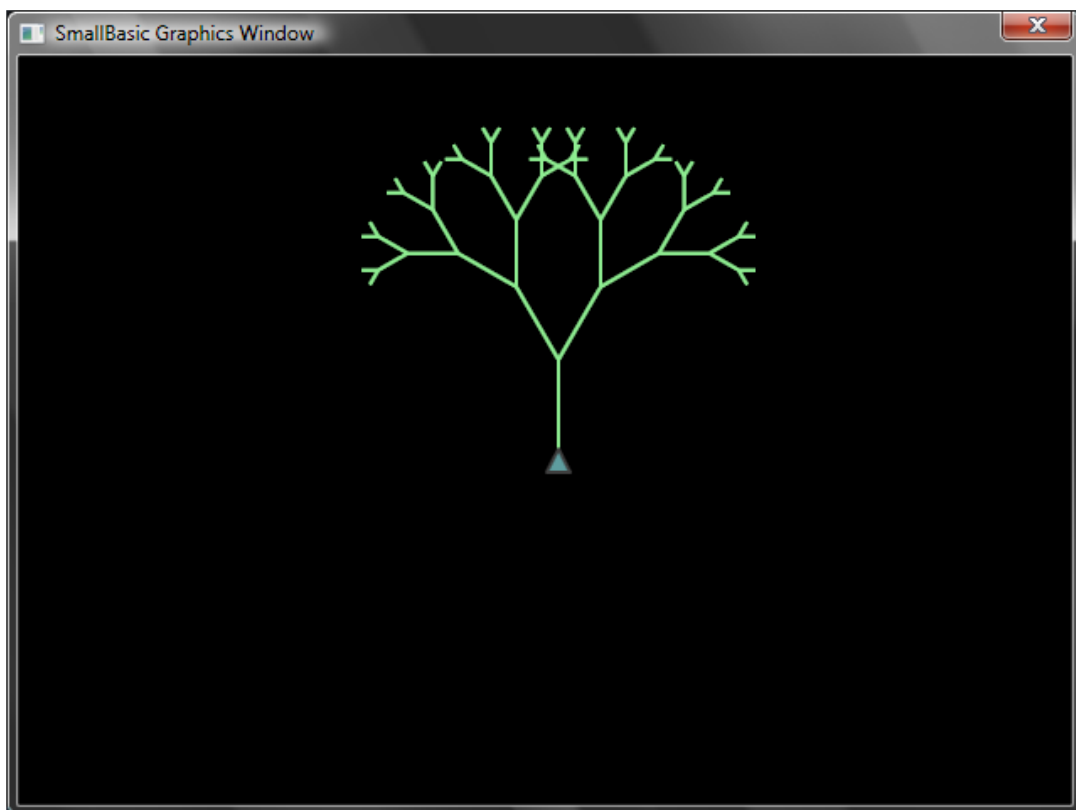
```
Sub OnMouseMove
    x = GraphicsWindow.MouseX
    y = GraphicsWindow.MouseY
    If (Mouse.IsLeftButtonDown) Then
        GraphicsWindow.DrawLine(prevX, prevY, x, y)
    EndIf
    prevX = x
    prevY = y
EndSub
```

Приложение А

## Забавни примери

---

### Фрактал с костенурка



Фигура 58: Костенурката чертае фрактал с форма на дърво

```
angle = 30  
delta = 10  
distance = 60  
Turtle.Speed = 9  
GraphicsWindow.BackgroundColor = "Black"
```

```
GraphicsWindow.PenColor = "LightGreen"  
DrawTree()
```

```
Sub DrawTree
```

```
  If (distance > 0) Then
```

```
    Turtle.Move(distance)
```

```
    Turtle.Turn(angle)
```

```
    Stack.PushValue("distance", distance)
```

```
    distance = distance - delta
```

```
    DrawTree()
```

```
    Turtle.Turn(-angle * 2)
```

```
    DrawTree()
```

```
    Turtle.Turn(angle)
```

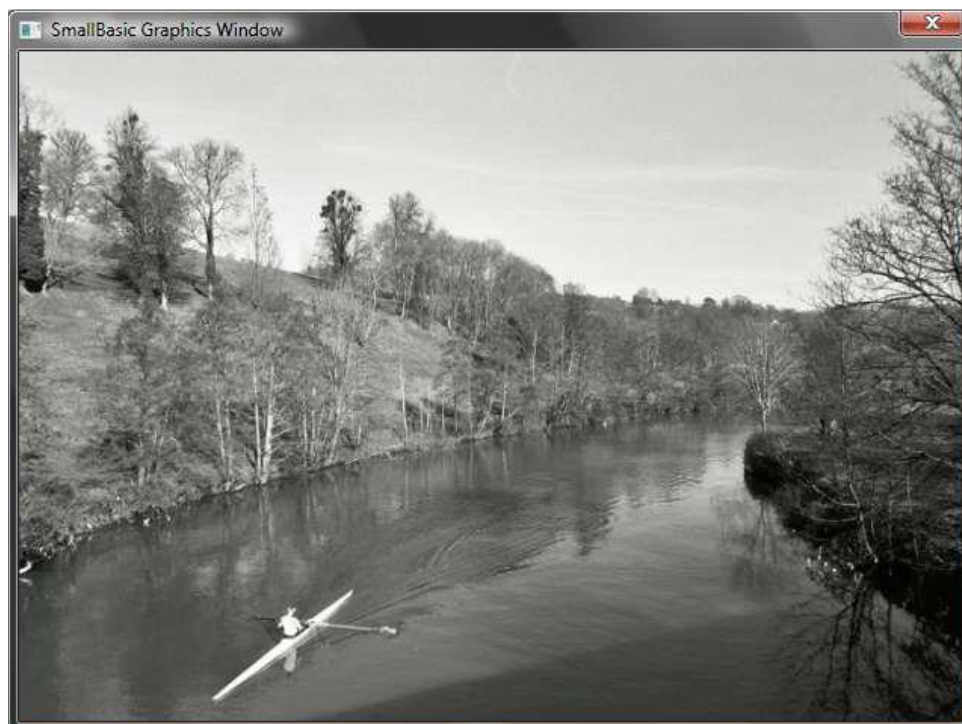
```
    distance = Stack.PopValue("distance")
```

```
  Turtle.Move(-distance)
```

```
EndIf
```

```
EndSub
```

## Снимки от Flickr



Фигура 59: Получаване снимки от Flickr

```
GraphicsWindow.BackgroundColor = "Black"
```

```
GraphicsWindow.MouseDown = OnMouseDown
```

```
Sub OnMouseDown
```

```
    pic = Flickr.GetRandomPicture("mountains, river")
```

```
    GraphicsWindow.DrawResizedImage(pic, 0, 0, 640, 480)
```

```
EndSub
```

## Динамичен тапет за десктопа

```
For i = 1 To 10
```

```
    pic = Flickr.GetRandomPicture("mountains")
```

```
    Desktop.SetWallPaper(pic)
```

```
    Program.Delay(10000)
```

```
EndFor
```

## Игра Paddle



Фигура 60: Игра Paddle

```
GraphicsWindow.BackgroundColor = "DarkBlue"
```

```
paddle = Shapes.AddRectangle(120, 12)
```

```
ball = Shapes.AddEllipse(16, 16)
```

```
GraphicsWindow.MouseMove = OnMouseMove
```

```

x = 0
y = 0
deltaX = 1
deltaY = 1

RunLoop:
    x = x + deltaX
    y = y + deltaY

    gw = GraphicsWindow.Width
    gh = GraphicsWindow.Height
    If (x >= gw - 16 or x <= 0) Then
        deltaX = -deltaX
    EndIf
    If (y <= 0) Then
        deltaY = -deltaY
    EndIf

    padX = Shapes.GetLeft (paddle)
    If (y = gh - 28 and x >= padX and x <= padX + 120) Then
        deltaY = -deltaY
    EndIf

    Shapes.Move(ball, x, y)
    Program.Delay(5)

    If (y < gh) Then
        Goto RunLoop
    EndIf

GraphicsWindow.ShowMessage("You Lose", "Paddle")

Sub OnMouseMove
    paddleX = GraphicsWindow.MouseX
    Shapes.Move(paddle, paddleX - 60, GraphicsWindow.Height - 12)
EndSub

```

## Приложение Б

# Цветовете

Ето списък с имена на цветовете поддържани от Small Basic, категоризирани на база оттенък.

### Red Colors

|             |         |
|-------------|---------|
| IndianRed   | #CD5C5C |
| LightCoral  | #F08080 |
| Salmon      | #FA8072 |
| DarkSalmon  | #E9967A |
| LightSalmon | #FFA07A |
| Crimson     | #DC143C |
| Red         | #FF0000 |
| FireBrick   | #B22222 |
| DarkRed     | #8B0000 |

### Pink Colors

|                 |         |
|-----------------|---------|
| Pink            | #FFC0CB |
| LightPink       | #FFB6C1 |
| HotPink         | #FF69B4 |
| DeepPink        | #FF1493 |
| MediumVioletRed | #C71585 |
| PaleVioletRed   | #DB7093 |

### Orange Colors

|             |         |
|-------------|---------|
| LightSalmon | #FFA07A |
| Coral       | #FF7F50 |
| Tomato      | #FF6347 |
| OrangeRed   | #FF4500 |
| DarkOrange  | #FF8C00 |
| Orange      | #FFA500 |

### Yellow Colors

|                      |         |
|----------------------|---------|
| Gold                 | #FFD700 |
| Yellow               | #FFFF00 |
| LightYellow          | #FFFFE0 |
| LemonChiffon         | #FFFACD |
| LightGoldenrodYellow | #FAFAD2 |
| PapayaWhip           | #FFEFD5 |
| Moccasin             | #FFE4B5 |
| PeachPuff            | #FFDAB9 |

|               |         |
|---------------|---------|
| PaleGoldenrod | #EEE8AA |
| Khaki         | #F0E68C |
| DarkKhaki     | #BDB76B |

## Purple Colors

|                 |         |
|-----------------|---------|
| Lavender        | #E6E6FA |
| Thistle         | #D8BFD8 |
| Plum            | #DDA0DD |
| Violet          | #EE82EE |
| Orchid          | #DA70D6 |
| Fuchsia         | #FF00FF |
| Magenta         | #FF00FF |
| MediumOrchid    | #BA55D3 |
| MediumPurple    | #9370DB |
| BlueViolet      | #8A2BE2 |
| DarkViolet      | #9400D3 |
| DarkOrchid      | #9932CC |
| DarkMagenta     | #8B008B |
| Purple          | #800080 |
| Indigo          | #4B0082 |
| SlateBlue       | #6A5ACD |
| DarkSlateBlue   | #483D8B |
| MediumSlateBlue | #7B68EE |

## Green Colors

|             |         |
|-------------|---------|
| GreenYellow | #ADFF2F |
| Chartreuse  | #7FFF00 |
| LawnGreen   | #7CFC00 |
| Lime        | #00FF00 |
| LimeGreen   | #32CD32 |
| PaleGreen   | #98FB98 |
| LightGreen  | #90EE90 |

|                   |         |
|-------------------|---------|
| MediumSpringGreen | #00FA9A |
| SpringGreen       | #00FF7F |
| MediumSeaGreen    | #3CB371 |
| SeaGreen          | #2E8B57 |
| ForestGreen       | #228B22 |
| Green             | #008000 |
| DarkGreen         | #006400 |
| YellowGreen       | #9ACD32 |
| OliveDrab         | #6B8E23 |
| Olive             | #808000 |
| DarkOliveGreen    | #556B2F |
| MediumAquamarine  | #66CDAA |
| DarkSeaGreen      | #8FBC8F |
| LightSeaGreen     | #20B2AA |
| DarkCyan          | #008B8B |
| Teal              | #008080 |

## Blue Colors

|                 |         |
|-----------------|---------|
| Aqua            | #00FFFF |
| Cyan            | #00FFFF |
| LightCyan       | #E0FFFF |
| PaleTurquoise   | #AFEEEE |
| Aquamarine      | #7FFFD4 |
| Turquoise       | #40E0D0 |
| MediumTurquoise | #48D1CC |
| DarkTurquoise   | #00CED1 |
| CadetBlue       | #5F9EA0 |
| SteelBlue       | #4682B4 |
| LightSteelBlue  | #B0C4DE |
| PowderBlue      | #B0E0E6 |
| LightBlue       | #ADD8E6 |
| SkyBlue         | #87CEEB |

|   |    |
|---|----|
| Предговор .....                         | 2  |
| Колектив.....                           | 3  |
| Въведение.....                          | 4  |
| Разбиране на нашата първа програма..... | 8  |
| Представяне на променливи.....          | 12 |
| Условия и разклонения.....              | 17 |
| Цикли.....                              | 23 |
| Графика за начинаещи .....              | 27 |
| Забавление с фигури .....               | 35 |
| Графика с костенурка .....              | 40 |
| Под-процедури.....                      | 48 |
| Масиви .....                            | 54 |
| Събития и интерактивност .....          | 61 |
| Забавни примери.....                    | 67 |
| Цветове.....                            | 71 |
| Съдържание.....                         | 73 |

Настоящото учебно пособие е авторски и оригинален превод от Английски език на ръководството по програмиране на Small Basic на компанията Microsoft. Разрешава се разпространението на книгата, ако това е с нетърговска цел при изричното позоваване на източника и авторите. За издаване на материала на хартиен носител или продажбата на електронни копия трябва да имате договор с авторите.

© Димитър Минчев, Мариан Панков

**Microsoft Small Basic. Въведение в програмирането.**

Бургаски Свободен Университет, 2015

ISBN 978-619-7126-03-7