# Ćwiczenia 1

Na pierwszych zajęciach wykorzystywane jest WSH (Windows Scripting Host), ćwiczenia są pisane w VBScript (Visual Basic Script Edition). WSH jest mechanizmem standardowo dostępnym w każdym systemie Microsoft Windows od wersji 2000. Do wersji starszych można go doinstalować ze strony Windows Update, jeśli go nie ma. Visual Basic jest językiem bardzo prostym, ma małe wymagania co do składni i nie trzeba określać typu definiowanych zmiennych – wystarcza do przetrenowania podstawowych technik programowania.

## Pierwsze zadanie

Uwaga! ze względu na pewne problemy jakie mogą wyniknąć, jeśli korzysta się w systemie z polskich czcionek, a przede wszystkim ze względu na różnice jakie mogą powstawać przy nieprawidłowych ustawieniach regionalnych (Regional Settings) w systemie, wykorzystywane będzie nazewnictwo angielskie. Generalnie ze względu na to, iż angielski jest 'informatycznym esperanto' i brakuje jednolitego sposobu tłumaczenia wyrazów, pozwolę sobie korzystać raczej z angielskich odpowiedników, starając się podawać poslkie propozycje tłumaczenia.

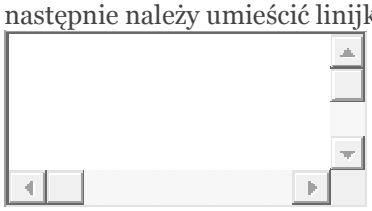
Pierwsze zadanie polega na przywitaniu się ze światem programistów – czyli standardowy program "hello world".

Należy:

- założyć plik "my first script.vbs"
  aby to zrobić należy w wybranym katalogu kliknąć **RMB** (*Right Mouse Button*), wybrać '**utwórz nowy dokument tekstowy**' i założyć plik o wybranej nazwie. Standardowo skonfigurowany Windows oszuka użytkownika zakładając plik "my first script.vbs.txt" – ponieważ 'txt' jest standardowym rozszerzeniem pliku tekstowego. Łatwo można to dostrzec, ponieważ pozostaje ikonka pliku 'txt': . Plik 'vbs' powinien mieć ikonkę symbolizującą skrypt: . Aby móc swobodnie oglądać i zmieniać rozszerzenia plików należy z okna explorera wybrać:
- '`narzędzia - opcje folderów - widok`' i odznaczyć opcję '`ukrywaj rozszerzenia znanych typów plików`'
  ('`tools - folder options - view`' '`hide extentions for known file types`')
- Wyedytować plik otwierając go w notatniku [*notepad*]

  można to zrobić np klikając **RMB** na utworzonym pliku i wybierając opcję '**edytuj**'. Wybranie opcji 'otwórz' jest próbą uruchomienia skryptu. Ponieważ jest on na razie pusty nie powinno się zupełnie nic stać [być może mignie okienko linii poleceń przez ułamek sekundy].

  Innym sposobem na edycję tego pliku jest uruchomienie notatnika, po czym wybranie z menu '`plik - otwórz`' i wybranie stworzonego pliku z odpowiedniego katalogu.
- następnie należy umieścić linijkę kodu o następującej treści:



```
1 WScript.echo "Hello World!!!"
```

- zachować zmiany

  '`ctrl-s`' lub '`plik - zapisz`'
- i uruchomienie programu
  zupełnie tak jak uruchamia się zwykły program w systemie – czyli klikając na nim myszką lub klawiszem 'enter'.

Na ekranie powinno pojawić się okienko z wybraną treścią:

# Błędy w programie

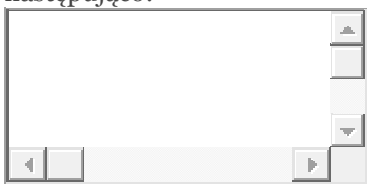
Jeśli w programie został popełniony jakiś błąd, pojawi się komunikat wyglądający podobnie do przedstawionego poniżej:

Należy przeczytać treść błędu!

Niestety powszechną manierą jest odruch jak najszybszego zamknięcia wyskakujących komunikatów. Pokutuje to setkami wirusów na komputerze oraz oczywistymi wynikającymi z tego powodu niejasnościami – brakiem świadomości, co się tak na prawdę stało. W przypadku pisania programów wielką sztuką okaże się umiejętność czytania błędów. W przypadku języków skryptowych jest to stosunkowo proste. W przedstawionym okienku wyraźnie określone jest miejsce wystąpienia błędu oraz jego opis:

- **Script** (*Skrypt*): pokazuje dokładne położenie pliku, w którym wystąpił błąd
- **Line** (*Linia*): określa linię w pliku skryptu, która wygenerowała błąd
- **Char** (*Znak*): pokazuje przypuszczalny znak (wiersz), w którym interpreter napotkał na błąd
- **Error** (*Błąd*): opis błędu
- **Code** (*Kod*): nr kodu błędu – nie każdy błąd musi mieć opis. Opis często jest zbyt enigmatyczny, aby pomógł. Poszukując w Internecie jakiegoś opisu na pewno trzeba będzie skorzystać z numeru błędu – jest on niezależny od języka i mamy pewność, że nie jest to inny błąd z podobnym opisem.
- **Source** (*źródło*): który moduł interpretera wygenerował błąd

Na przykładzie wyraźnie widać, że błąd wystąpił w pierwszej linii, 27 znaku a opis mówi ‘nie zakończony String’. Zmienna typu ‘**String**‘ oznacza ‘ciąg znaków’ – o typach zmiennych będzie za chwilkę. Kod błędnego programu wyglądał następująco:



```
1 WScript.echo "Hello World!
```

Jak widać zabrakło na końcu linijki apostrofu ". Zwykły tekst jest potraktowany jak zmienna typu**String**. Zilustruje to błędny kod w następnym przykładzie, aby pokazać, iż wszystkie wartości, jakie są umieszczone w kodzie mogą być reprezentowane przez **Zmienne**.

# Zmienne

Zmienne są jednym z najistotniejszych elementów w większości języków programowania. Za ich pomocą w łatwy sposób można automatyzować zadania – co będzie wyraźnie widoczne przy wykorzystaniu pętli.

Zmienne służą do czasowego przechowywania różnych wartości – np. stringów (ciągów znaków), wartości liczbowych i innych. Spójrzmy na poniższy kod, prezentujący przypisanie wartości do zmiennej:

```
1 mojaZmienna = "Hello World!"
2 WScript.echo mojaZmienna
3 mojaZmienna = "bye bye"
4 WScript.echo mojaZmienna
```

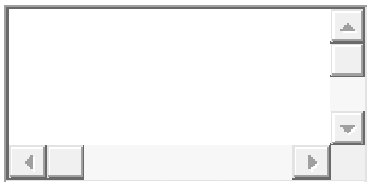
W (1) linii została zdefiniowana zmienna o nazwie ‘mojaZmienna’ o wartości “Hello World!”. (2) linijka taki właśnie napis powinna wyświetlić na ekranie.

Pomimo, że w (4) linijce ponownie wyświetlamy wartość tej samej zmiennej na ekranie pojawi się już napis “bye bye”, ponieważ w (3) linijce zostało określone, iż przechowuje ona już inną wartość. Analogią do świata rzeczywistego mogą być pudełka, do których coś się wrzuca. kiedy tylko potrzebujemy tego czegoś – wystarczy, że znamy nazwę pudełka.

### Zadanie

Przetestuj zadanie, tworząc kolejny skrypt i umieszczając powyżej przedstawioną treść.

Na zmiennych można wykonywać różne operacje:



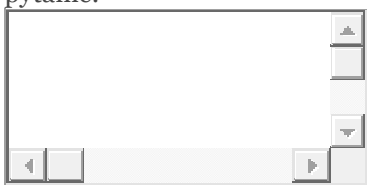
```
1 liczba1 = 2
2 liczba2 = 3
3 wscript.echo liczba1+liczba2
```

### Zadanie

Utwórz kolejny skrypt z przedstawioną treścią. Jaka wartość zostanie wyświetlona? Dlaczego?

## Interakcja i Funkcje

Język VBS nie został stworzony z myślą o interakcji z użytkownikiem, niemniej posiada prymitywne mechanizmy, pozwalające dostarczyć wartości w trakcie działania skryptu – czyli takie, które nie są znane podczas uruchomienia. Taką możliwość daje funkcja ‘**inputBox**‘. Funkcja ta wymaga co najmniej jednego parametru – tekstu, który się wyświetli jako pytanie:



```
1 mojaZmienna = inputBox("podaj imie:")
2 wscript.echo mojaZmienna
```

Funkcje są to operacje zwracające jakąś wartość. W tym przykładzie zastosowana została funkcja ‘inputBox’ (1), która wyświetla okienko z pytaniem i polem umożliwiającym wpisanie jakiegoś tekstu. Tekst ten będzie wartością, jaką ta funkcja zwróci w momencie zakończenia swojego działania – tutaj po kliknięciu guzika ‘ok’.

W skrypcie wartość ta zostanie zachowana w zmiennej ‘mojaZmienna’. Dzięki temu (2) linijka wyświetli tekst wpisany przez użytkownika.

Abstrakcyjnym przykładami funkcji z życia codziennego mogą być zwykłe operacje matematyczne – np. funkcja dodawania miałaby taki zapis:

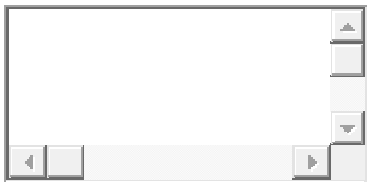
```
1 wynik = dodawaj(liczba1, liczba2)
```

Widać, że funkcja ‘dodawaj’ przyjmuje dwa parametry, i że zwraca jakąś wartość, która tutaj zostanie zapisana w zmiennej ‘wynik’.

## Pierwszy algorytm

Spróbujmy wykorzystać wiedzę z wykładu (na temat tego, czym jest ‘algorytm’) oraz z pierwszych ćwiczeń o zmiennych, i stwórzmy pierwszy program. Zadaniem będzie napisanie skryptu, który będzie wyświetlał kolejno liczby od 1 do 10. W pierwszym kroku należy wymyślić sposób działania programu – czyli algorytm.

Pierwszym nasuwającym się rozwiązaniem byłoby stworzenie takiego skryptu:



```
1 wscript.echo "1"
2 wscript.echo "2"
3 [...]
4 wscript.echo "10"
```

co nawet przy 10 liczbach jest już tak nudne, że wolałem umieścić ‘[...]‘ niż je kolejno wypisywać. Wystarczy zmienić treść zadania: wypisz liczby od 1 do 50000, żeby zauważyć, że taki sposób nie jest dobry. Kluczem w takim zadaniu jest wykorzystanie zmiennej. Tak przedstawia się wstępny algorytm:

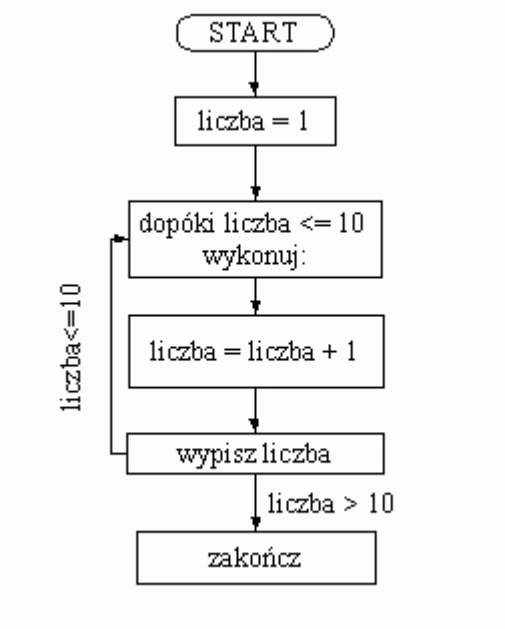
- przypisz zmiennej wartość początkową = 1
- wyświetl zmienną
- zwiększ wartość zmiennej o jeden
- jeśli wartość jest mniejsza od 10 – wróć do punktu (2)
- zakończ działanie

Należy teraz zamienić ten algorytm tak, aby był zapisany w pseudokodzie – czyli w języku bardzo zbliżonym do tego, w którym piszemy. Ułatwia to w następnych krokach stworzenie gotowego programu. W przyszłości wszystkie algorytmy powinny być zapisywane w pseudokodzie lub jako schemat blokowy [gdzie też opisy powinny być w pseudokodzie].

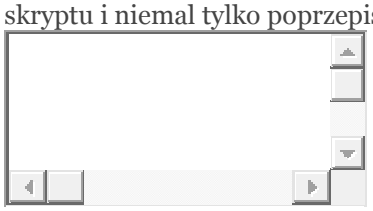
Dokonamy też pewnej modyfikacji – pokazującej jak działa pętla:

- liczba = 1
- dopóki liczba <= 10 póty wykonuj:
- wypisz liczba
- liczba = liczba +1
- wyświetl “zakończono sukcesem”

Trik z zastąpieniem warunku na pętlę ‘póty póki’ zagwarantuje nam skok do odpowiedniej linii. Z poprzedniego algorytmu nie było jeszcze widać jakiż to mechanizm spowoduje, że program zacznie znów wykonywać żądany blok programu. Działanie programu ładnie wizualizuje ten sam algorytm przedstawiony w postaci blokowej:




Odpowiednikiem ‘póki-póty’ w VBS jest pętla ‘Do While ... loop’. Z tymi informacjami, wystarczy stworzyć nowy plik skryptu i niemal tylko poprzepisywać dane z algorytmu:



```
1 liczba =1
2 do while liczba &lt;=10
3     wscript.echo liczba
4     liczba = liczba + 1
5 loop
6 wscript.echo "zakończono sukcesem"
```

blok programu wykonywanego w pętli to wszystko to, co znajduje się od linijki tuż po warunku ['**do while**' - czyli 'wykonuj póki'], aż do momentu aż program natrafi na słówko ‘**loop**‘ czyli ‘*zapętl*‘. Wtedy działanie programu skacze znów do linii warunku, jest on sprawdzany, i jeśli jest prawdziwy to znów wykona się blok programu. Jeśli nie jest spełniony [liczba 11 nie jest <= 10] – program przejdzie do wykonywania od linii, następującej po słówku ‘**loop**‘ – czyli tutaj po prostu wyświetli komunikat “zakończono sukcesem”.

## Zadanie

Wykonaj przedstawiony skrypt. Przerób go w taki sposób, aby wartość początkowa i końcowa były podawane przez użytkownika. Czyli po uruchomieniu programu powinniśmy być poproszeni o pierwszą wartość oraz wartość kończącą, które zostaną wykorzystane dalej w skrypcie.