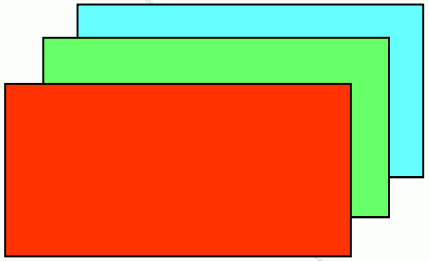
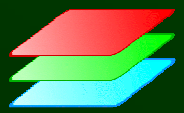


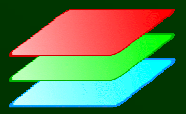
Języki skryptowe w programie Plans

Tomasz Zdun



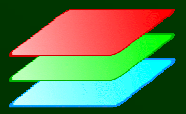
Zalety skryptów

- Automatyzacja powtarzających się czynności
- Rozszerzenie możliwości programu
  - Budowa własnych algorytmów analizy
  - Współpraca z innymi systemami(np. MS Office, system SCADA)
  - Współpraca wielu instancji programu Plans
  - itd...



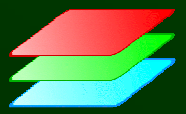
Języki skryptowe w Plansie

- język makropoleceń JMP
- VBScript
- JScript



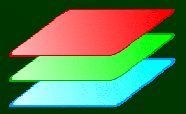
Język makropoleceń JMP

- Stworzony na potrzeby programu Plans
- Dostęp do wszystkich funkcji programu Plans
- Ograniczona funkcjonalność:
  - brak tablic
  - brak zmiennych lokalnych
  - ograniczona możliwość przekazywania wartości do procedur
  - brak funkcji



JScript i VBScript

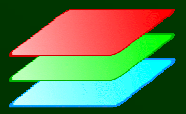
- Języki „wbudowane” do systemu Windows
- Wykorzystywane w wielu aplikacjach oraz na stronach WWW
- Bogata składnia:
  - pętle i instrukcje warunkowe
  - tablice i obiekty
  - funkcje
  - obsługa wyjątków
- Bogata dokumentacja



VBscript

- Oparty na języku BASIC
- Składnia podobna do Visual Basic for Application (pakiet MS Office), ale NIE taka sama

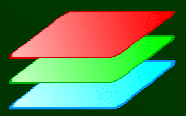
```
dim i, j
for i=1 to 10
  j =  j + i
next
cprintf(j)
```

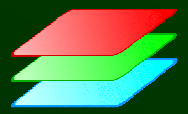


JScript

- Wersja języka JavaScript opracowana przez Microsft (Internet Explorer)
- Składnia podobna do języków Java, C i C++

```
var i, j=0;
for(i=1;i<10;i++)
{
  j += i;
}
cprintf(j);
```





SKŁADNIA JĘZYKA JSCRIPT. PODSTAWY

Podstawowe zasady

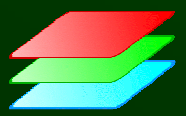
- Instrukcje oddzielone średnikiem lub końcem wiersza (elastyczność)
- Ważna wielkość liter
- Komentarz w jednej linii:

```
//komentarz
```

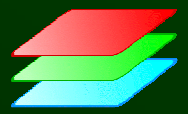
- Komentarz w wielu linii:

```
/* komentarz
kometarz */
```

- Blok kilku instrukcje wzięty w nawiasy klamrowe {}
- Wyświetlenie tekstu na okno: funkcja cprintf(text)



ZMIENNE



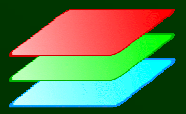
Deklaracja i typ

- Pierwsze użycie jest również deklaracją zmiennej

```
x = 10;
```

- Brak typów zmiennych

```
x = 10;
x = "Zmienna tekstowa";
x = new Array();
```



Zakres ważności

- Zmienne globalne:

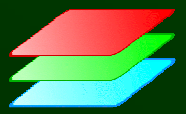
```
x = 10;
var x = 10;
```

- Zmienne lokalne funkcji - var:

```
function f1()
{
  var x = "tekst";
}
```

- Zalecane deklarowanie **wszystkich** zmiennych ze słowem

var



Konwersja typów

- Typ numeryczny → tekst
  - konwersja automatyczna

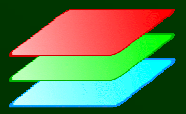
    ```
    cprintf("Liczba" + 20);
    ```
  - konwersja jawna

    ```
    cprintf("Liczba" + x.toString(10));
    ```

- Tekst → typ numeryczny
  - konwersja jawna do liczby całkowitej

    ```
    x = 10 + parseInt("20");
    ```
  - do liczby zmiennoprzecinkowej

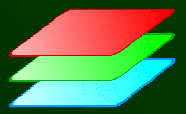
    ```
    x = 10.8 + parseFloat("20.9");
    ```



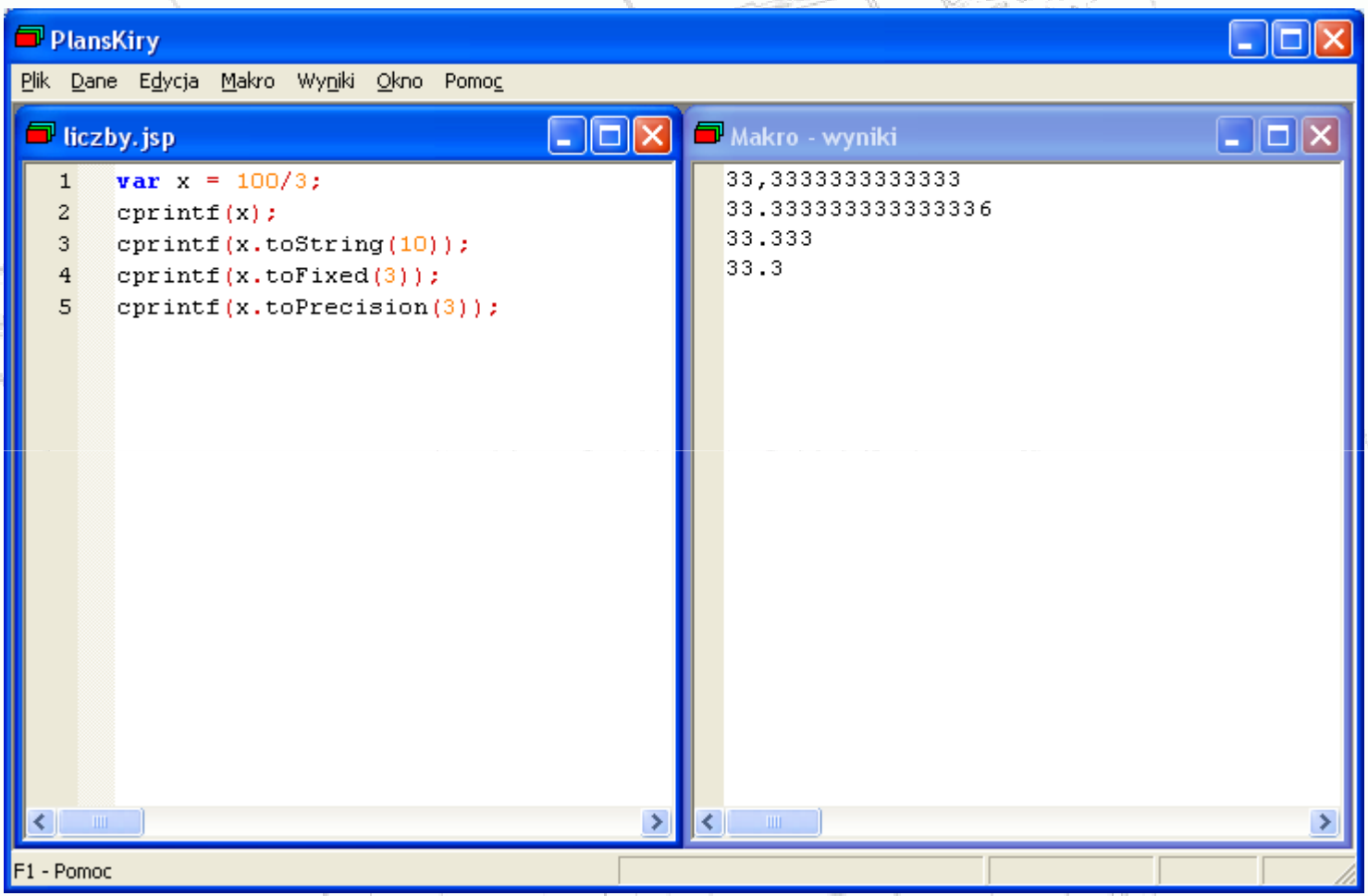
Formatowanie liczb rzeczywistych

```
var x = 100/3;
```

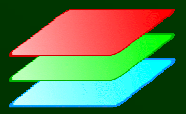
- Konwersja do tekstu:
  `x.toString(10)`
  (10-system dziesiętny, 2-system binarny itp.)
- Zadana liczba cyfr po kropce
  `x.toString(ile)`
- Zadana liczba cyfr znaczących (przed i po kropce)
  `x.toFixed(ile)`
- Zadana dokładność w postaci długość.precyzja:
  brak



Formatowanie liczb rzeczywistych

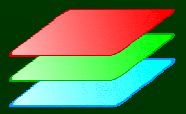


INSTRUKCJE STERUJĄCE



Wyrażenia logiczne

Prawda	true
Fałsz	false
Porównania:	==, !=, <, >, >=, <=
Iloczyn logiczna (and)	&&
Suma logiczna (or)	||
Negacja	!



Wyrażenia logiczne

- x w przedziale (10,20)

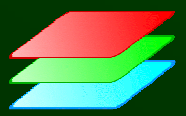
```
X>10 && x<20
```

- x z poza przedziału (10,20)

```
X<=10 || x>=20
```

- x w przedziale (0,10) lub (20,30)

```
(x>0 && x<10) || (x>20 && x<30)
```



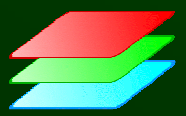
Instrukcja warunkowa if

- Postać 1

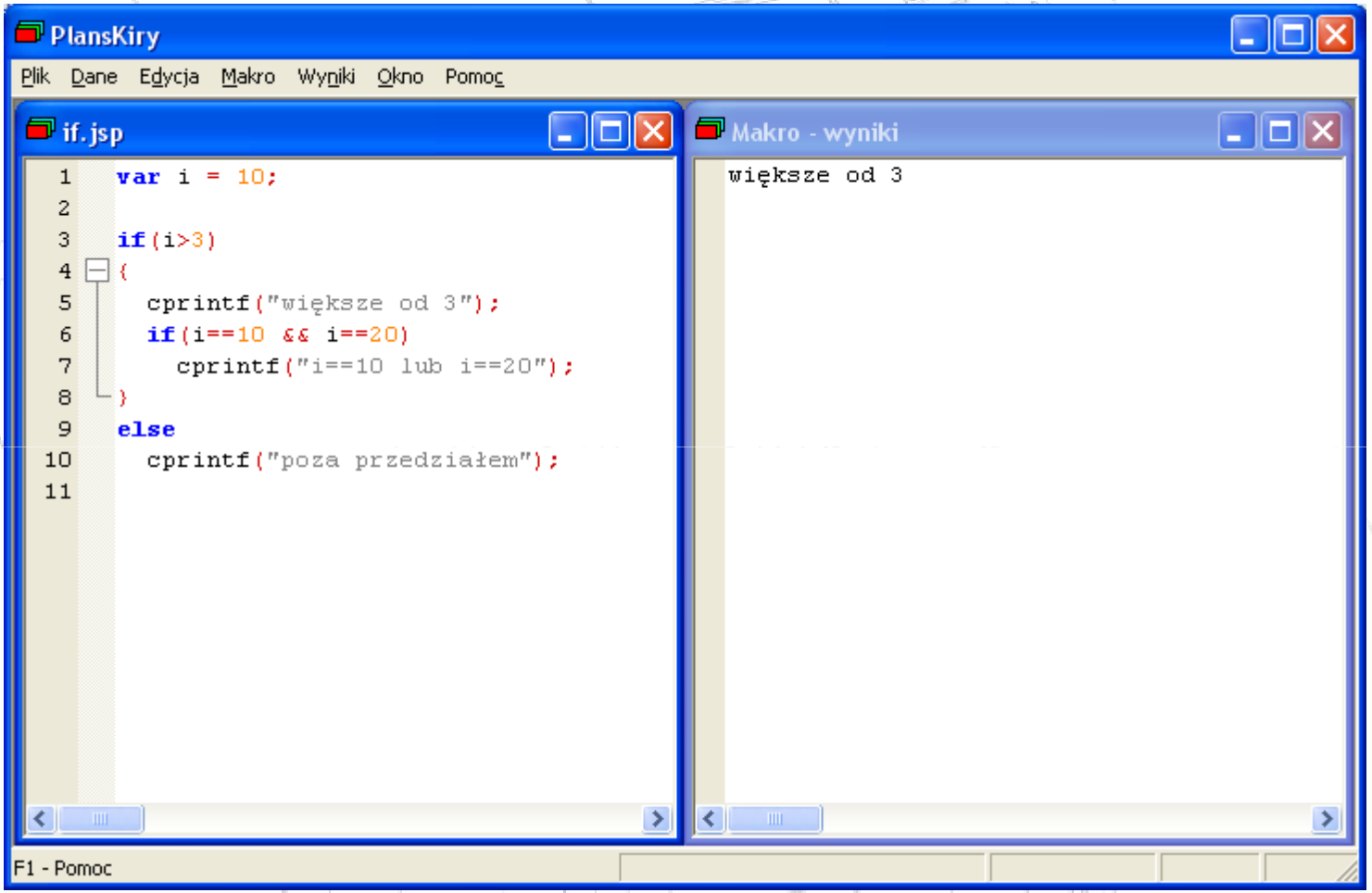
```
if(warunek)instrukcja1;
else instrukcja2;
```

- Postać 2

```
if(warunek)
{
  instrukcja1;
  instrukcja2;
}
else ...
```

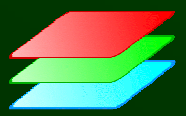


Instrukcja warunkowa if

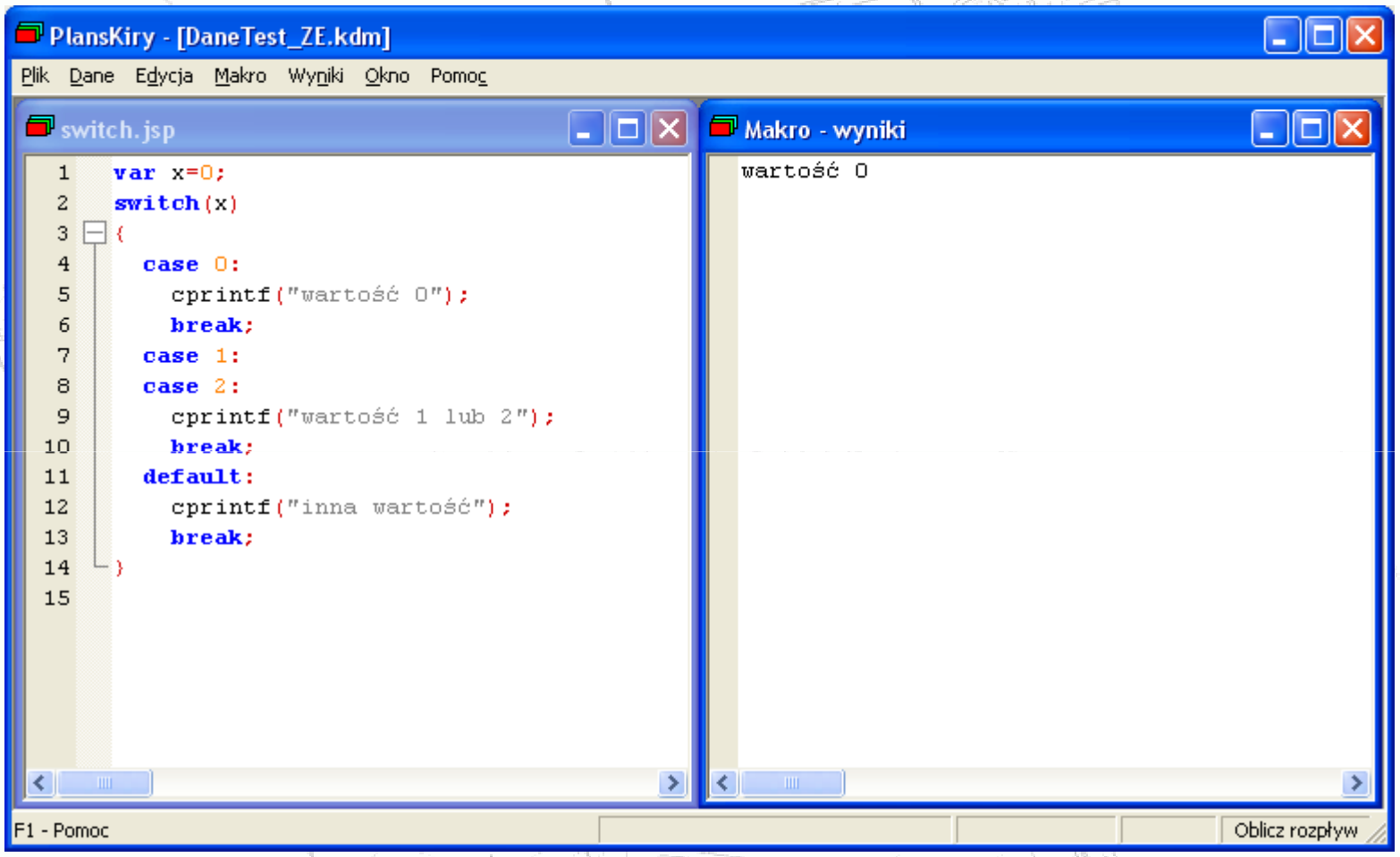


Instrukcja wyboru switch

```
switch(wyrażenie){
    case wartość1:
        instrukcje1;
        break;
    case wartość2:
    case wartość4:
        instrukcja2_3;
        break;
    default:
        instrukcje3;
        break;
}
```



Instrukcja wyboru switch



Pętla for

```
for(inicjalizacja; warunek; inkrementacja)
{
        instrukcja1;
        instrukcja2;
        ...
        }
```

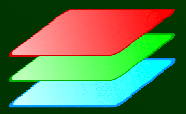
Inkrementacja

Zwiększenie licznika *i* o jeden

```
i = i+1  <-> i+=1 <-> i++
```

Zmniejszenie licznika *i* o jeden

```
i = i-1  <-> i-=1 <-> i--
```

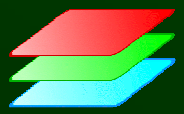


Pętla for


PlansKiry - [DaneTest_ZE.kdm]
Plik Dane Edycja Makro Wyniki Okno Pomoc
for.jsp
1 for(i=1; i<=10; i++)
2 {
3 cprintf(i+ '...');
4 }
5
6 for(i=10; i>0; i--)
7 {
8 cprintf('...'+i);
9 }
10
11
Makro - wyniki
1...
2...
3...
4...
5...
6...
7...
8...
9...
10...
...10
...9
...8
...7
...6
...5
...4
...3
...2
...1
F1 - Pomoc
X=273, Y=76
Krok=12

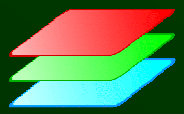
Pętla while

```
while(warunek)
{
        instrukcja1;
        instrukcja2;
        ...
}
```

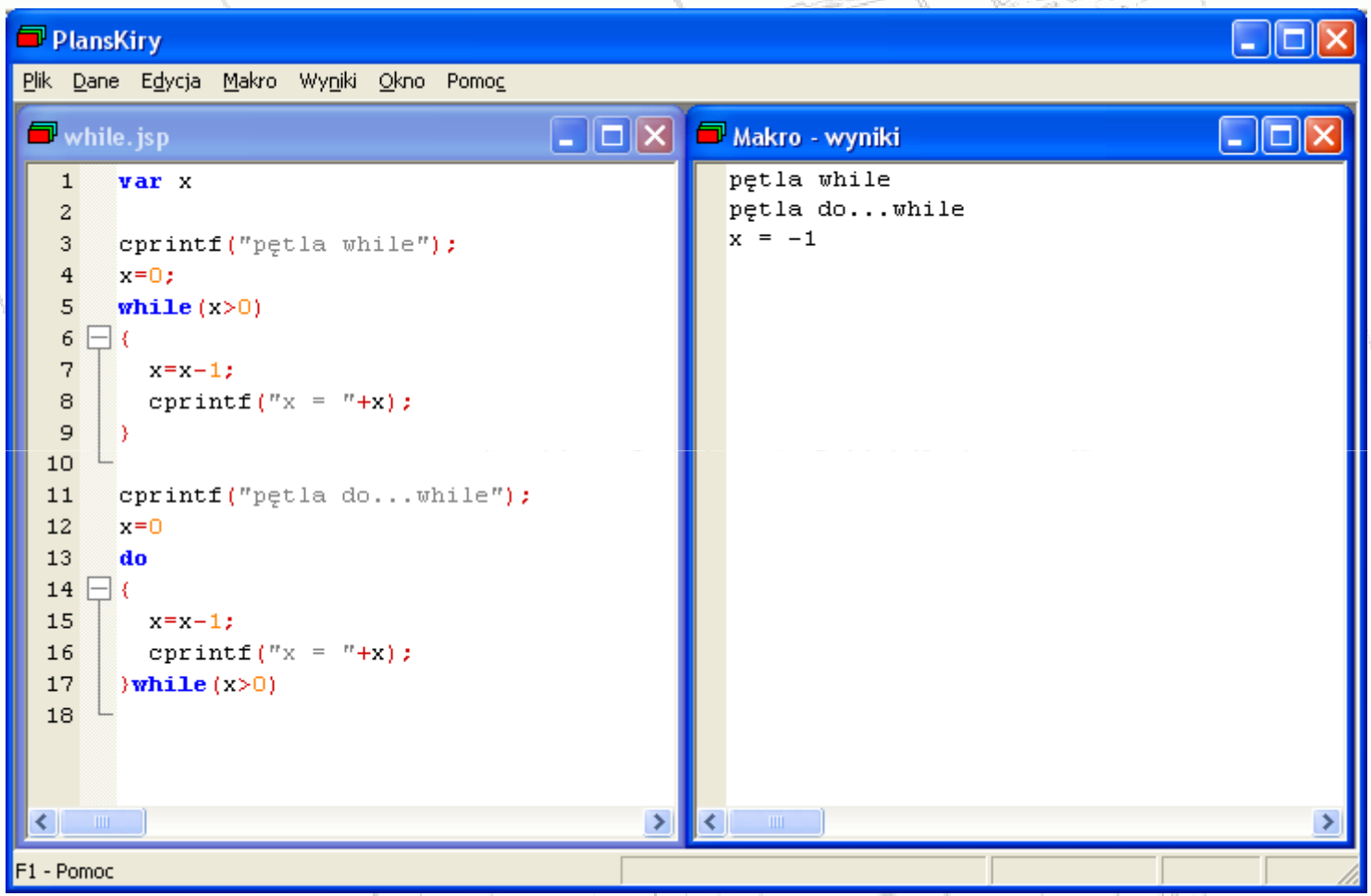


Pętla do…while

```
do
{
        instrukcja1;
        instrukcja2;
        ...
}
while(warunek)
```

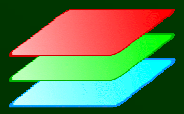


Pętla while oraz do…while

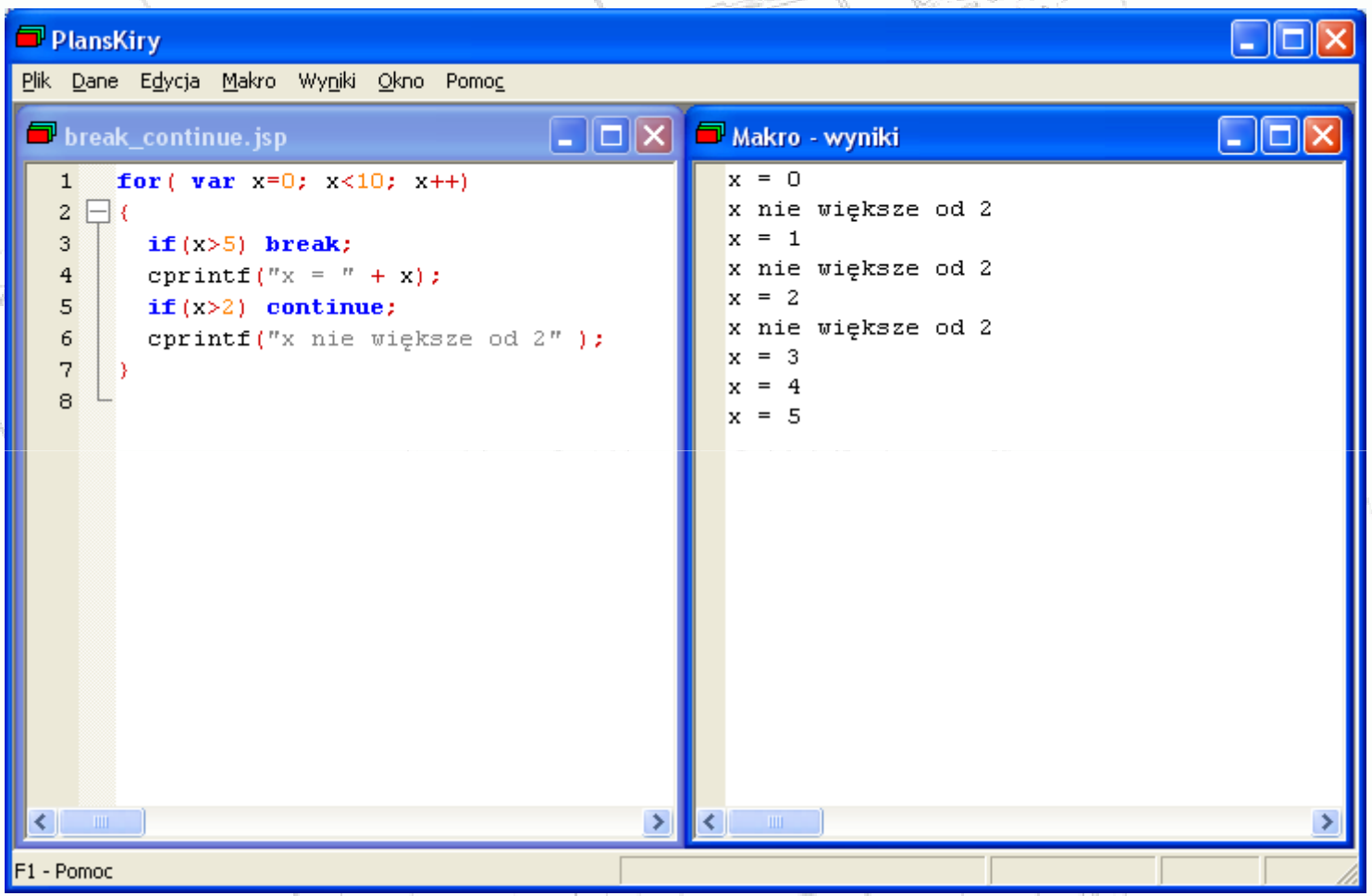


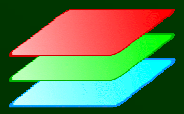
Instrukcje break i continue

- **break** – przerwanie wykonywania pętli
- **continue** – pominięcie dalszych instrukcji i sprawdzenie warunku pętli

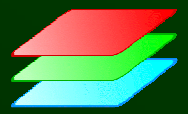


Instrukcje break i continue



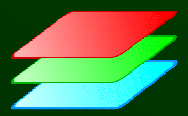


FUNKCJE



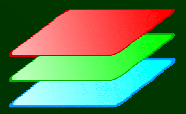
Definicje funkcji

```
function nazwa_funkcji(arg1, arg2,...)
{
        instrukcja1;
        instrukcja2;
        ...
        return(wartość);
}
```



Definicje funkcji - przykład

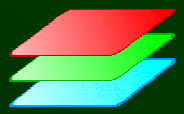
```
function SumaX2Y2(x, y)
{
	var z;
	z = x*x + y*y;
	return(z);
}
```



Wywołanie funkcji

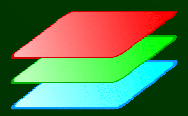
```
nazwa_funkcji(wartość1, wartość2,...)
```

`wartość1 ...` - przekazywane parametry do funkcji

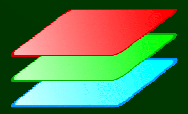


Wywołanie funkcji

```
function SumaX2Y2(x, y)
{
      var z;
      z = x*x + y*y;
      return(z);
}
var g = 20;
var h = SumaX2Y2(5,g);
```



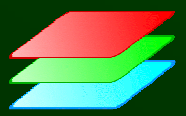
PRACA Z PLIKAMI



Obiekt FileSystemObject (FSO)

- Obiekt FSO jest częścią języka JScript
- Obiekt FSO udostępnia wiele metod do zarządzania folderami i plikami
- Dostęp do metod: *obiekt.metoda*
- Tworzenie obiektu FSO

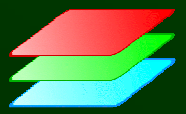
```
var fso = new ActiveXObject(
    "Scripting.FileSystemObject");
```



Otwarcie pliku

```
var plik = fso.OpenTextFile( "nazwa.txt",
    tryb, tworzyc );
```

- fso – utworzony obiekt FSO
- tryb:
  - 1 – do odczytu
  - 2 – do zapisu
  - 3 – do dopisywania na końcu pliku
- tworzyc – czy utworzyć plik, jeśli nie istnieje (true/false, domyślnie false)



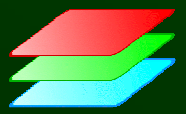
Zapis

- Zapis tekstu

```
plik.Write(tekst);
```

- Zapis tekstu i znaku końca linii

```
plik.WriteLine(tekst);
```



Odczyt

- Odczyt wiersza

```
var s = plik.ReadLine();
```

- Odczyt zadanej liczby znaków

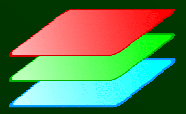
```
var s = plik.Read(ile_znakow);
```

- Odczyt całości

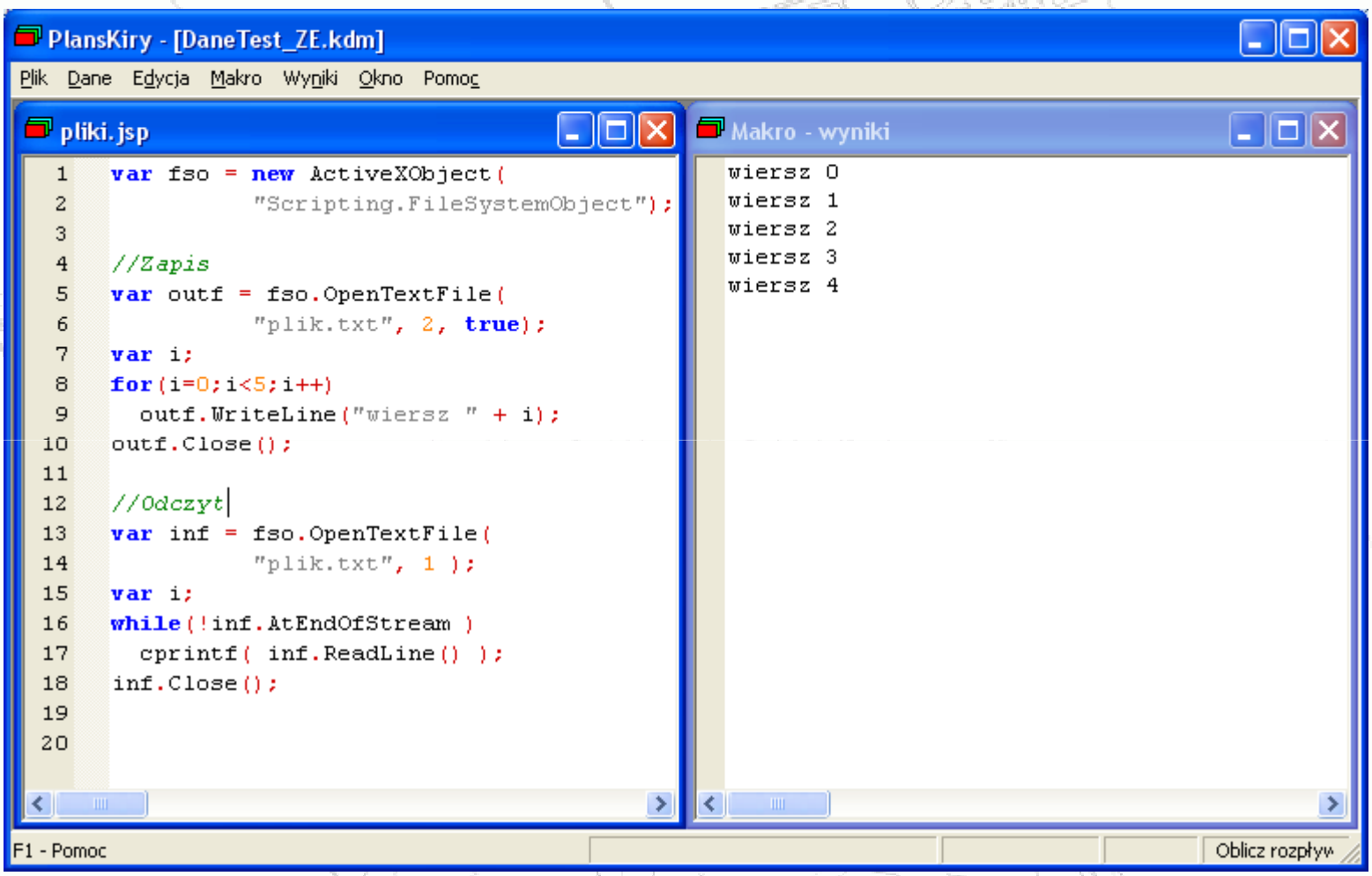
```
var s = plik.ReadAll();
```

- Test końca pliku

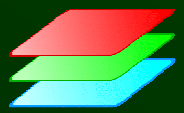
```
plik.AtEndOfStream
```



Zapis i odczyt



OBIEKTY (STRUKTURY)



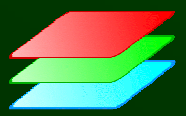
Struktury

- Złożony typ danych, np.:
- Grupuje dane różnego typu, ale logicznie powiązane

- Składa się z **pól**
- Pola definiuje użytkownik

- Przykład:

```
Generator
        Sn,
        Pmax,
        Pmin,
        ...
```



Obiekty w JScript

- Tworzenie obiektu:

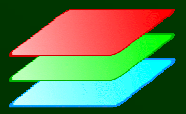
```
var gen = new Object();
```

- Dodawanie pól:

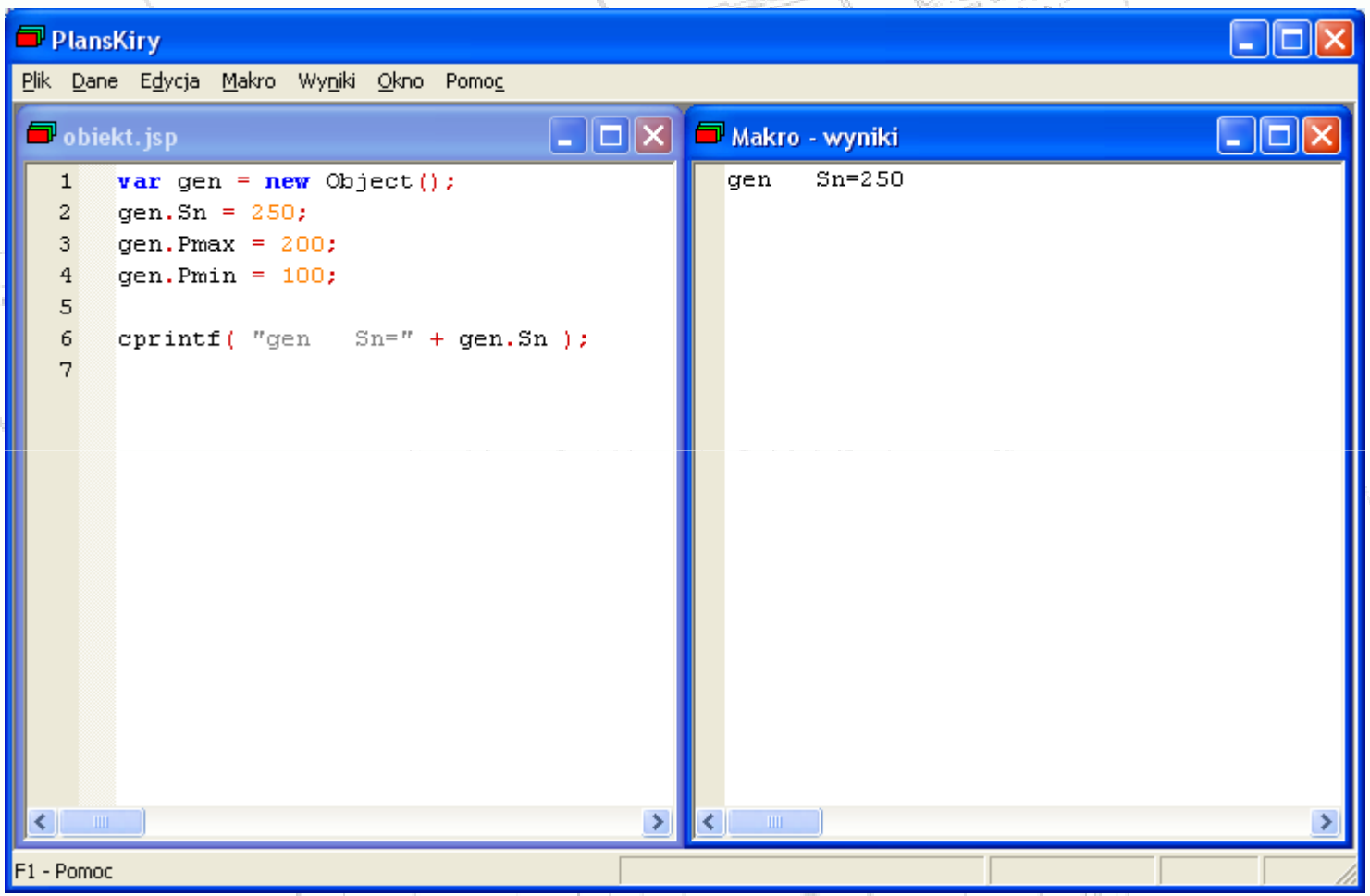
```
gen.Sn = 250;
gen.Pmax = 200;
```

- Odczyt wartości:

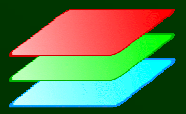
```
cprintf("Sn = " + Sn );
```



Obiekty


PlansKiry
Plik Dane Edycja Makro Wyniki Okno Pomoc
obiekt.jsp
1 var gen = new Object();
2 gen.Sn = 250;
3 gen.Pmax = 200;
4 gen.Pmin = 100;
5
6 cprintf("gen Sn=" + gen.Sn);
7
Makro - wyniki
gen Sn=250
F1 - Pomoc

TABLICE



Tworzenie tablic

- Tworzenie pustej tablicy:

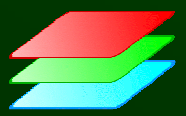
```
var tab1 = new Array();
```

- Tworzenie tablicy o zadanym rozmiarze:

```
var tab2 = new Array(rozmiar);
```

- Tworzenie wypełnionej tablicy

```
var tab3 = new Array(el1,el2,el3 ...);
```



Wstawianie wartości

- Dodanie wartości na koniec

  `tab.push(wartosc)`

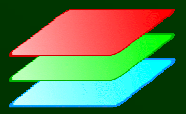
- Zmiana wartości

  `tab[indeks]=wartosc;`

  (jeżeli `indeks<rozmiar`, to rozmiar tablicy zostanie automatycznie zwiększony)

- Łączenie tablic

  `tab.concat(tab1, tab2, tab3);`



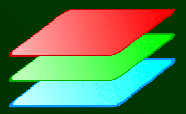
Dostęp do wartości

- Rozmiar tablicy:

```
tab.length
```

- Dostęp do wartości:

```
var z = tab[indeks];
```



Przydatne funkcje

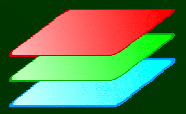
- Sortowanie

```
tab.sort();
tab.sort( funkcja_sortująca );
```

- Konwersja na tekst

```
tab.join( separator );
```

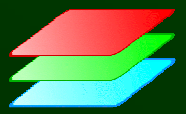
Separatorem może być znak końca linii: "\n"



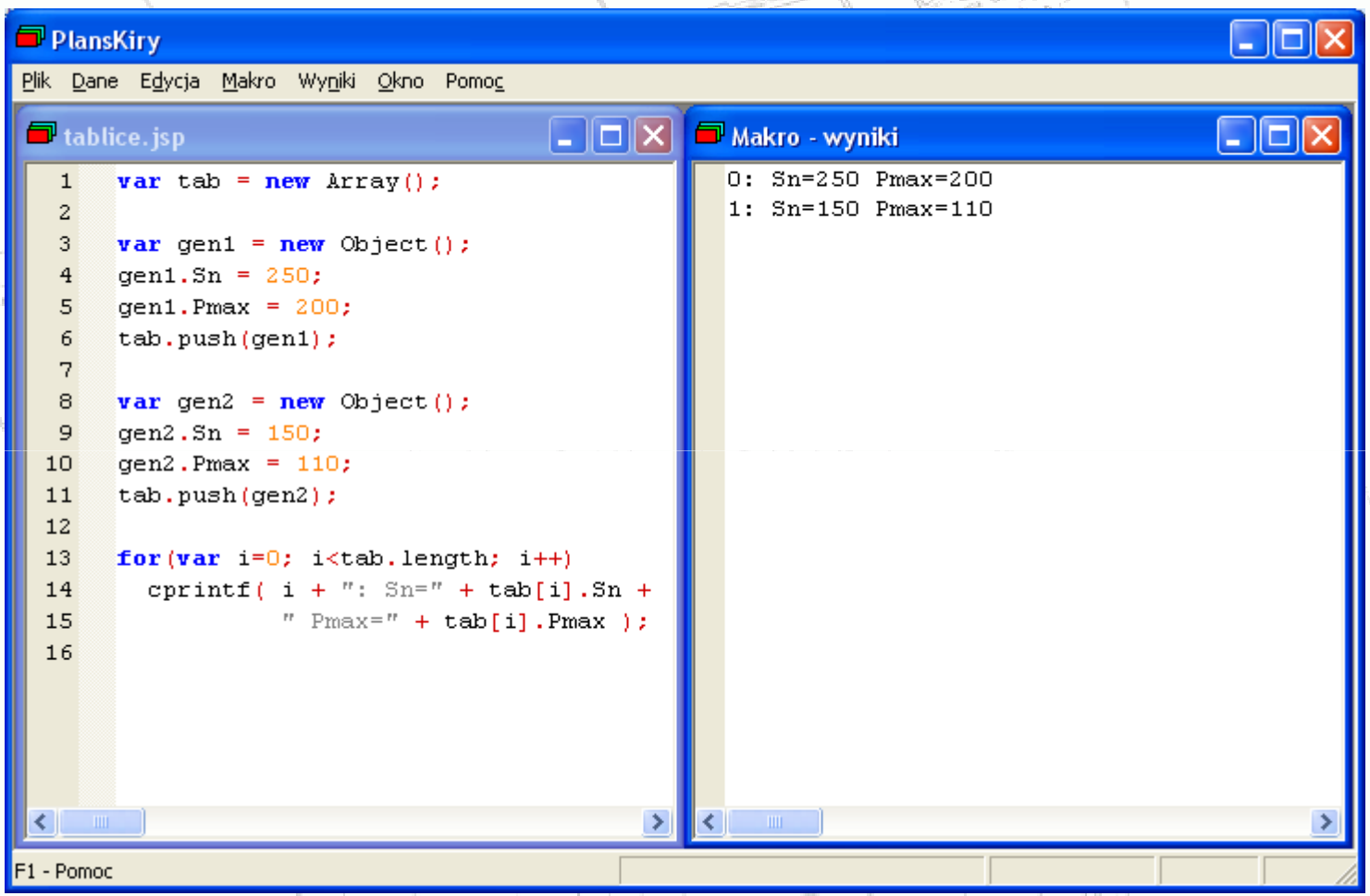
Tablica struktur (obiektów)

- Dodanie do tablicy obiektu

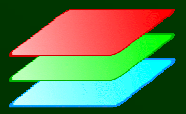
```
var tab = new Array();

var gen1 = new Object();
gen1.Sn = 250;
gen1.Pmax = 200;
tab.push(gen1);

var gen2 = new Object();
gen2.Sn = 150;
gen2.Pmax = 110;
tab.push(gen2);
```



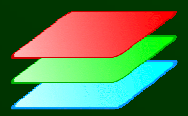
Tablica struktur (obiektów)



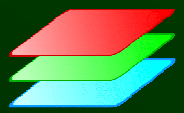
FUNKCJE UDOSTĘPNIONE PRZEZ PROGRAM PLANS



Nazwa	Opis
AddDataKDM	Doczytanie sieci do aktualnego modelu
CalcLF	Obliczenie rozpływu mocy
cprintf	Wypisanie komunikatu na ekranie
GetFile	Odszukanie pliku na dysku
ReadDataBIN	Wczytanie danych z pliku binarnego
ReadDataEPC	Wczytanie danych z pliku EPC
ReadDataKDM	Wczytanie danych z pliku KDM
ReadDataIEN	Wczytanie danych z pliku IEN
ReadDataUZP	Doczytanie uzupełnień
ReadDataUCTE	Wczytanie danych z pliku UCTE
SaveDataBIN	Zapisanie modelu do pliku binarnego
SaveDataEPC	Zapisanie modelu do pliku EPC
SaveDataIEN	Zapisanie modelu do pliku IEN
SaveDataKDM	Zapisanie modelu do pliku KDM
SaveDataUCTE	Zapisanie modelu do pliku UCTE
TestConnection	Test spójności
TestRX	Test R/X

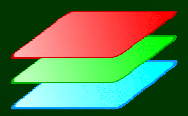


DOSTĘP DO DANYCH SIECIOWYCH



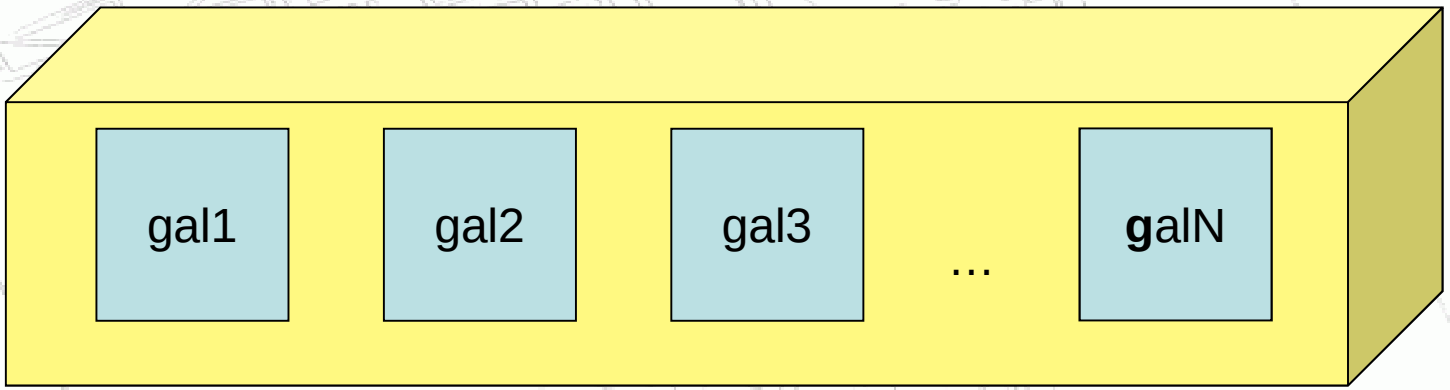
Udostępnione tablice i struktury

Nazwa	Opis
BraArray	Tablica gałęzi
Calc	Struktura zawierająca parametry procesu obliczeniowego
Data	Struktura zawierająca informacje o modelu
GenArray	Tablica generatoró
LinArray	Tablica linii
LodArray	Tablica odbiorów
Main	Struktura zawierająca informacje o programie
NodArray	Tablica węzłów
Rslt	Struktura zawierająca informacje o wyniku procesu obliczania rozpływu mocy
TrfArray	Tablica transformatorów
ZonArray	Tablica obszarów

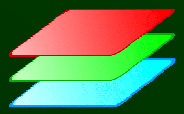


Tablica gałęzi

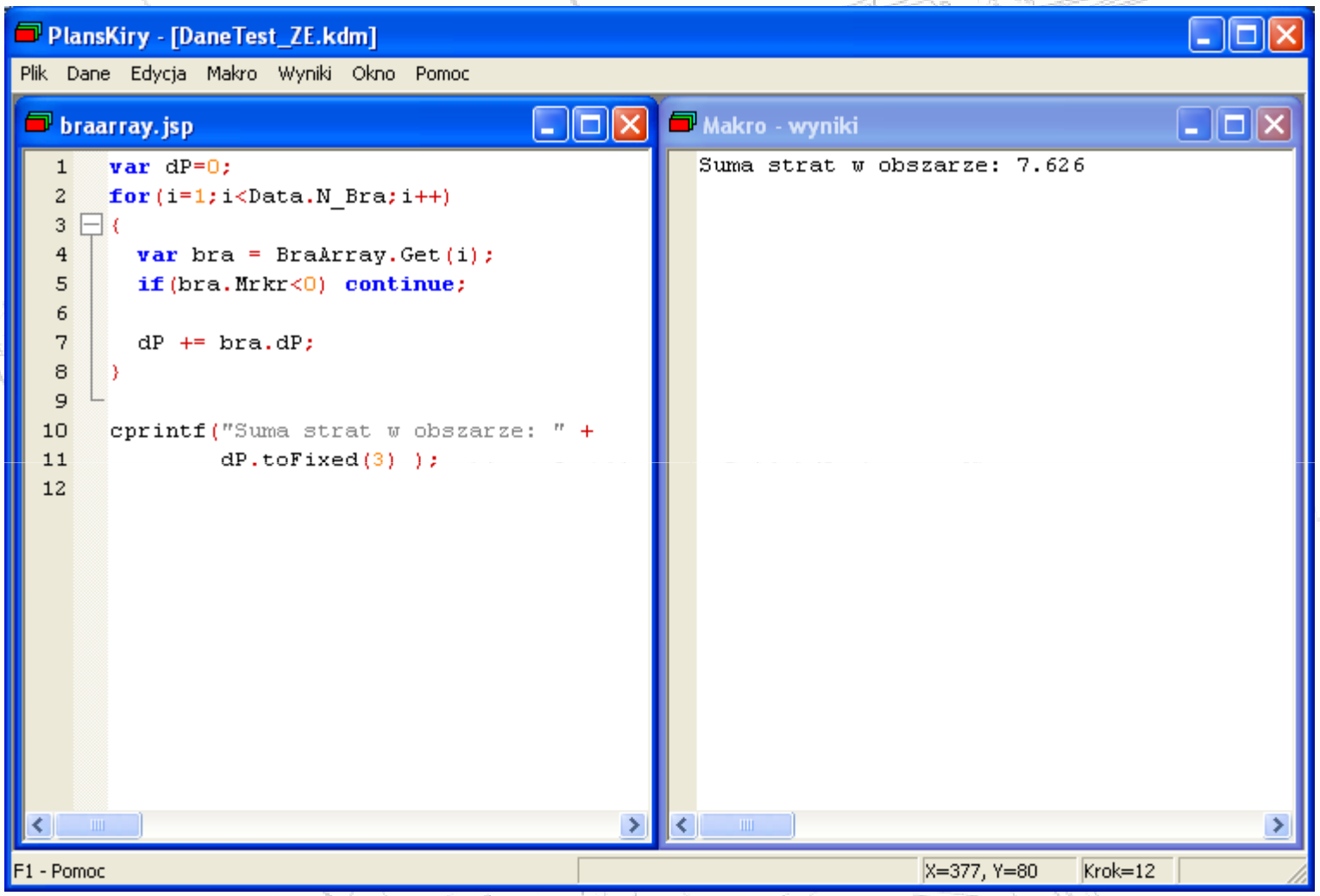
- Tablica gałęzi BraArray zawiera wszystkie gałęzie.



- Dostęp do gałęzi poprzez metody Get i Find



Tablica gałęzi



Jedna gałąź

```
var galaz1 = BraArray.Get(indeks);
var galaz2 = BraArray.Find("L10");
```

- Nazwa:

```
cprintf(galaz1.Name);
```

- Reaktancja:

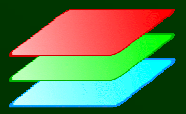
```
cprintf(galaz1.Xb.tofixed(1));
```

- Status:

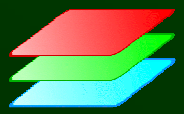
```
cprintf(galaz1.St);
```

- Wyłączenie gałęzi:

```
galaz1.St=-1;
```



PLANS JAKO SERWER AUTOMATYZACJI



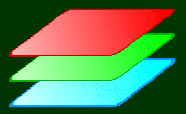
Uruchomienie drugiego Plansa

- Uruchomienie:

```
var plans2 = new ActiveXObject(
    "PlansEMC.Application");
```

- Praca:

```
plans2.ReadDataKDM("");
plans2.CalcLF("");
```



Excel w Plansie

- Uruchomienie:

```
var ex = new ActiveXObject(
   "Excel.Application");
ex.visible = true; //excel widoczny
```

- Utworzenie nowego arkuszu

```
ex.Workbooks.Add();
```

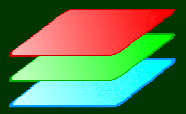
- Dostęp do komórek w arkuszu

```
ex.Cells(1, 1).Value = 123.4;
```

- Zapis arkusza i zamknięcie Excela

```
ex.Save();
ex.Quit();
```

- Więcej w dokumentacji Excela



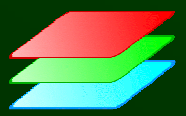
Plans z zewnątrz

- Uruchomienie ze skryptu zewnętrznego (plik *plans2_z_zewnatrz.js*):

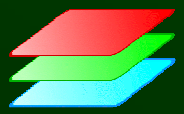
```
var plans = new ActiveXObject(
   "PlansEMC.Application");
```

- Praca z Plansem:

```
plans.ReadDataKDM("");
plans.CalcLF();
plans.BraArray.Find("C006„).St=-1;
plans.CalcLF();
...
```



DOKUMENTACJA



Dokumentacja

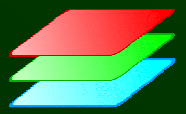
- Funkcje programu Plans:

  plik ***MakraJS.chm***

- Strony WWW
  - http://msdn.microsoft.com

    http://msdn.microsoft.com/en-us/library/ff729665(v=VS.94).aspx

  - http://www.poradnik-webmastera.com/kursy/javascript/



PRZYKŁADY WYKORZYSTANIA SKRYPTÓW