

# Przetwarzanie danych w SAS

[sgh.piotr.rozenbajgier@outlook.com](mailto:sgh.piotr.rozenbajgier@outlook.com)

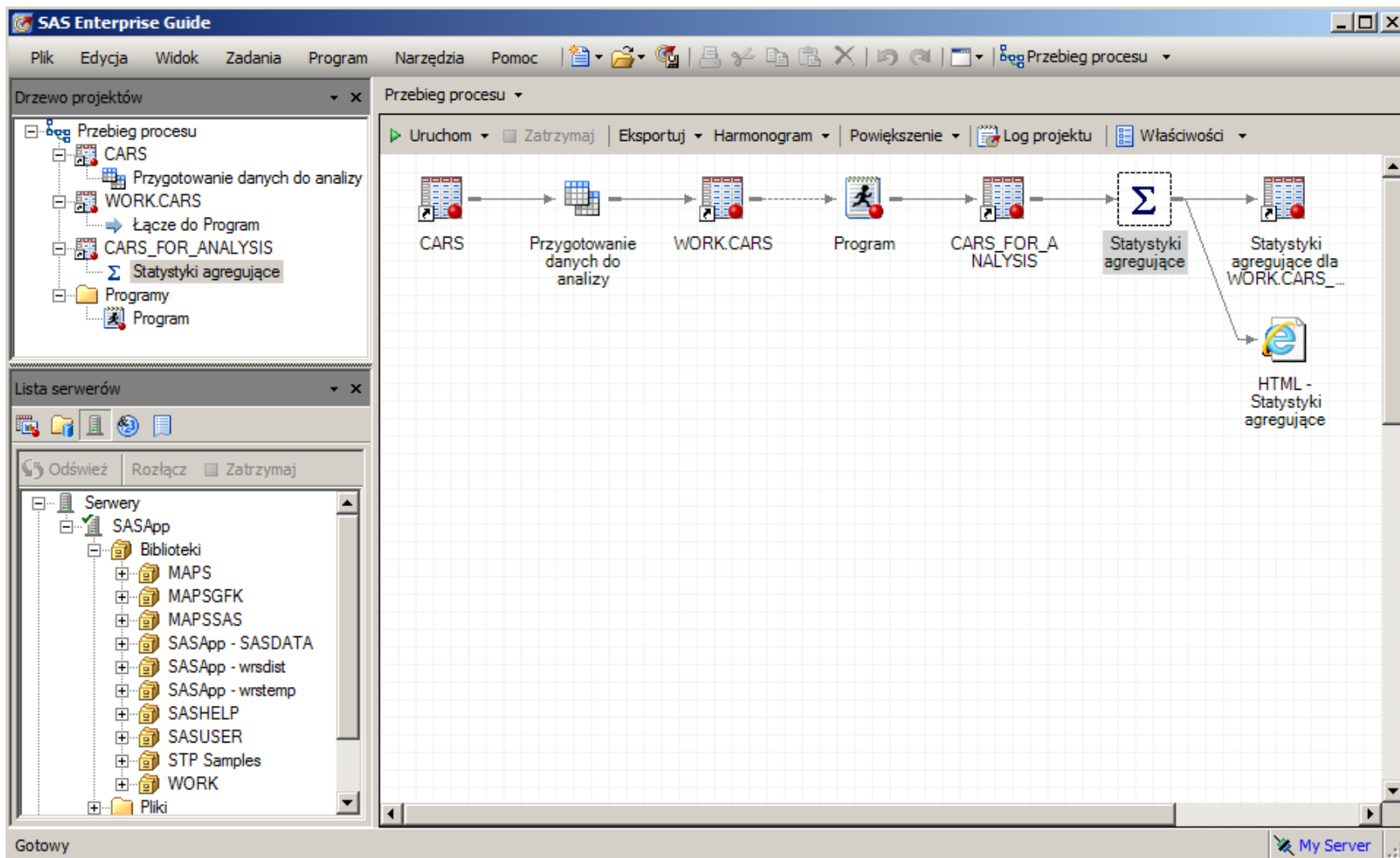
# Agenda cz. 1

- Środowisko pracy
- Sposób zapisu i przechowywania danych
- Wprowadzenie do języka SAS 4GL
  - Zbiory danych
  - Data step
  - Opcje zbiorów
  - Funkcje
  - Tablice
  - Pętle

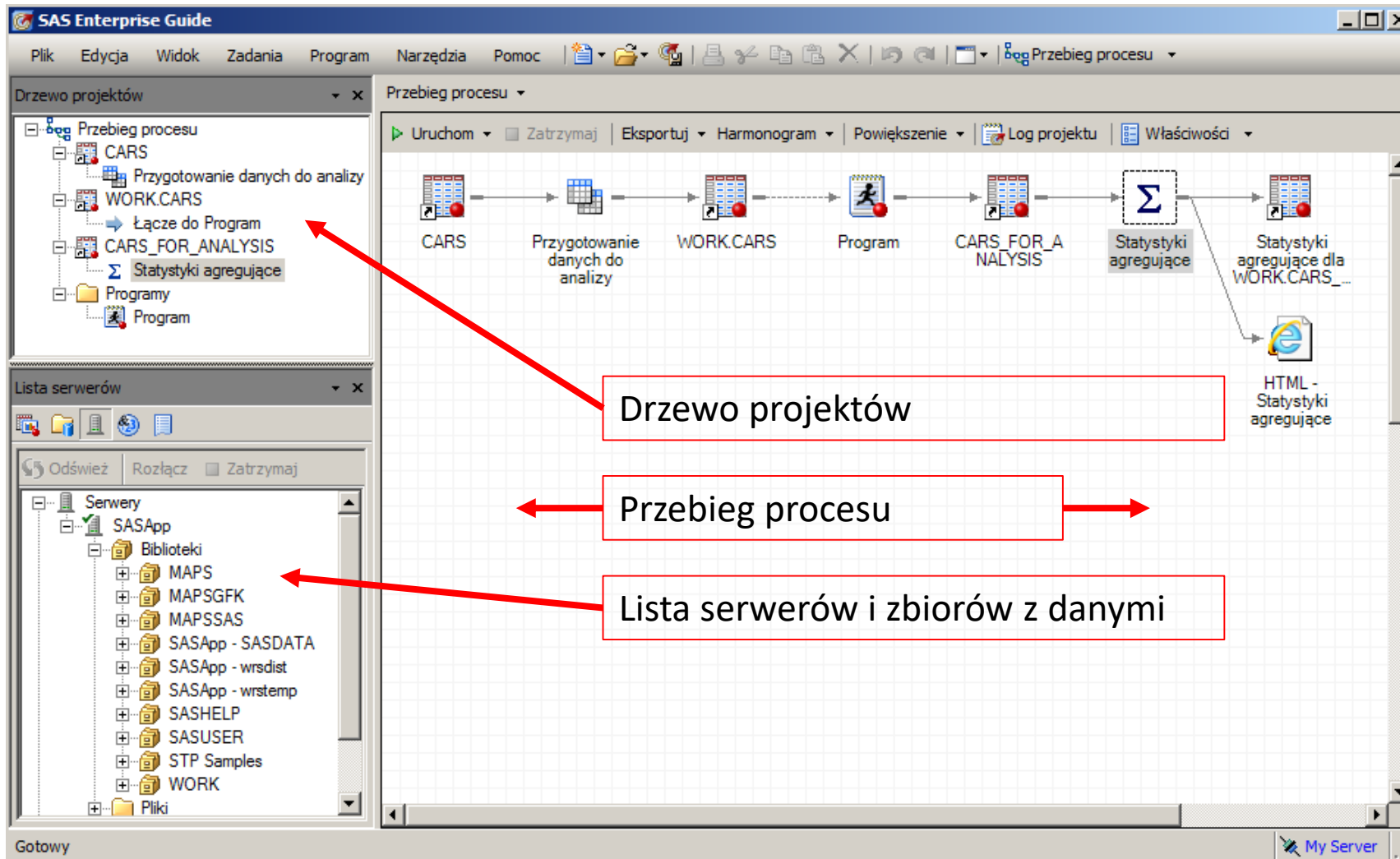
# Środowisko pracy

- „Podstawowe” środowiska pracy analityka
  - SAS BASE / SAS Foundation / SAS Display Manager
  - SAS Enterprise Guide
- „Zaawansowane” środowiska pracy analityka
  - SAS Enterprise Miner
  - SAS Forecast Studio

# Środowisko pracy



# Środowisko pracy



# SAS Enterprise Guide – okno powitalne

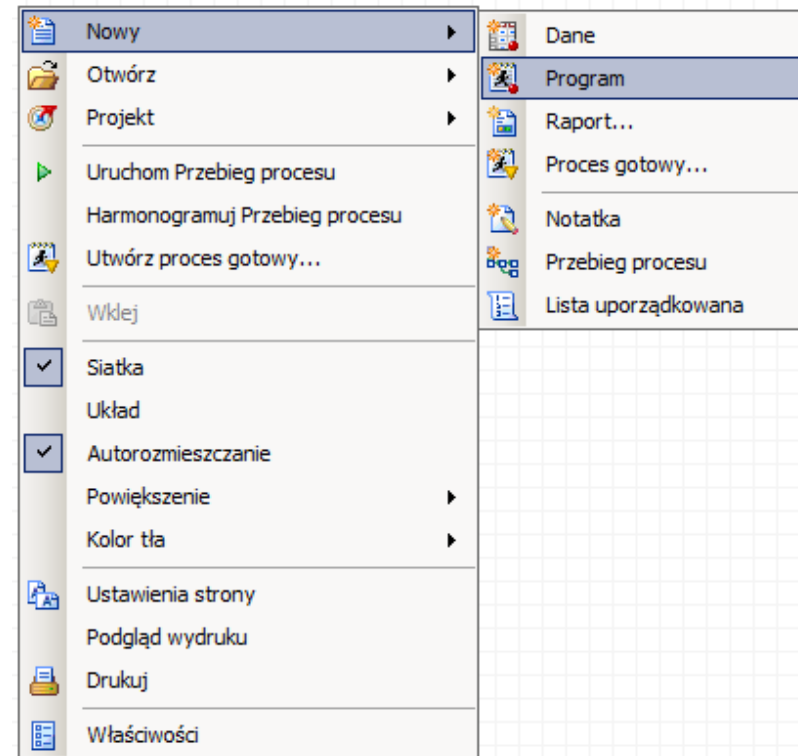
Po uruchomieniu aplikacji SAS Enterprise Guide widoczne jest okno powitalne, które umożliwia stworzenie nowego projektu lub otwarcie ostatnio używanego projektu.



# SAS Enterprise Guide – tworzenie nowego programu

Nowy program możemy stworzyć na kilka sposobów:

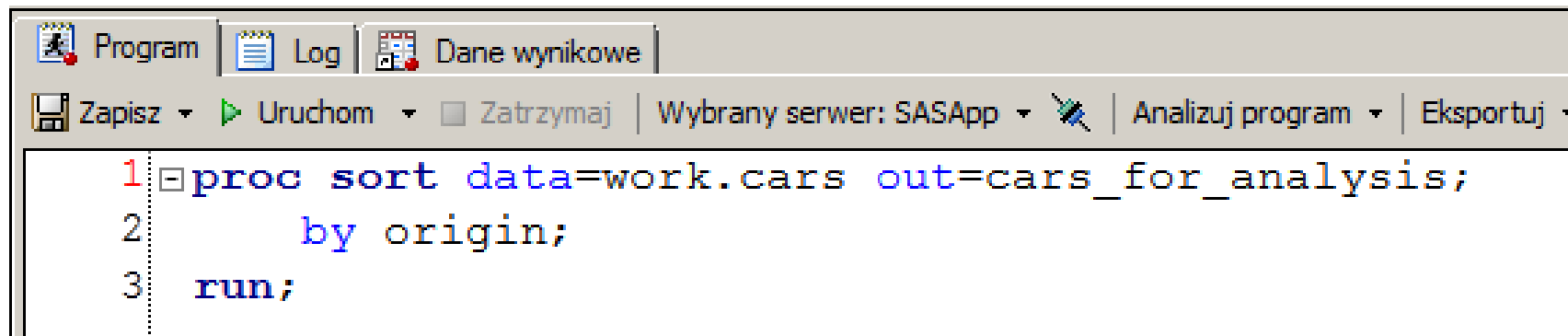
- Z menu podręcznego dla przebiegu procesu: **Nowy->Program**
- Z menu głównego aplikacji: **Plik->Nowy->Program**



# Enhanced Editor

**Enhanced Editor** jest edytorem kodu, który:

- umożliwia uruchamianie kodu,
- koloruje składnię,
- podpowiada składnię.



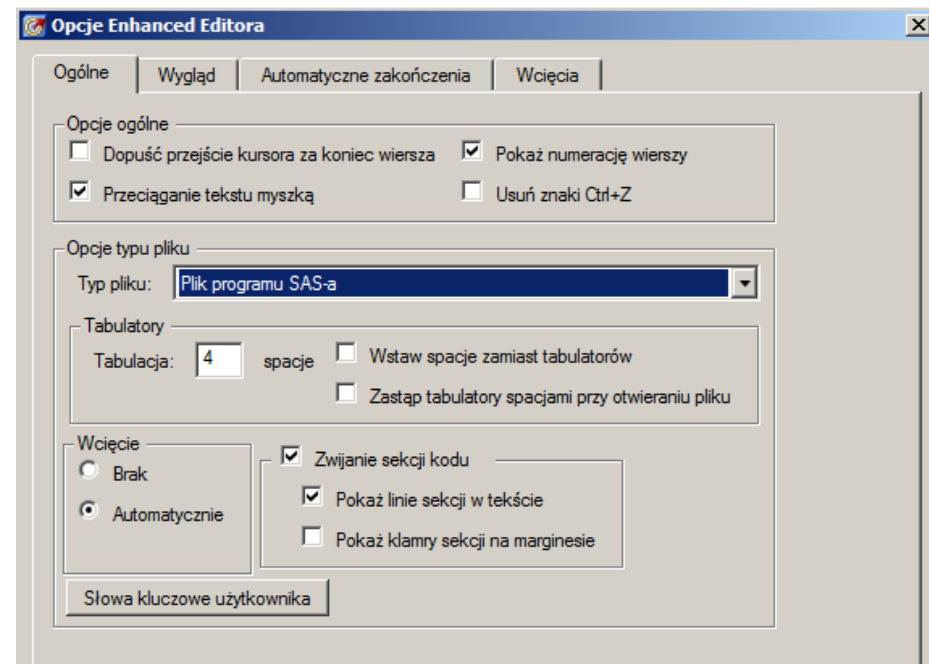
The screenshot displays the SAS Enhanced Editor window. The title bar at the top contains three tabs: 'Program' (active), 'Log', and 'Dane wynikowe'. Below the tabs is a toolbar with icons and labels for 'Zapisz' (Save), 'Uruchom' (Run), 'Zatrzymaj' (Stop), 'Wybrany serwer: SASApp' (Selected server: SASApp), 'Analizuj program' (Analyze program), and 'Eksportuj' (Export). The main editing area shows a SAS program with three lines of code, each preceded by a line number (1, 2, 3) in the left margin. The code is color-coded: 'proc' is red, 'sort' is blue, 'data=' is black, 'work.cars' is black, 'out=' is blue, 'cars\_for\_analysis;' is black, 'by' is blue, 'origin;' is black, and 'run;' is blue.

```
1 proc sort data=work.cars out=cars_for_analysis;  
2     by origin;  
3 run;
```



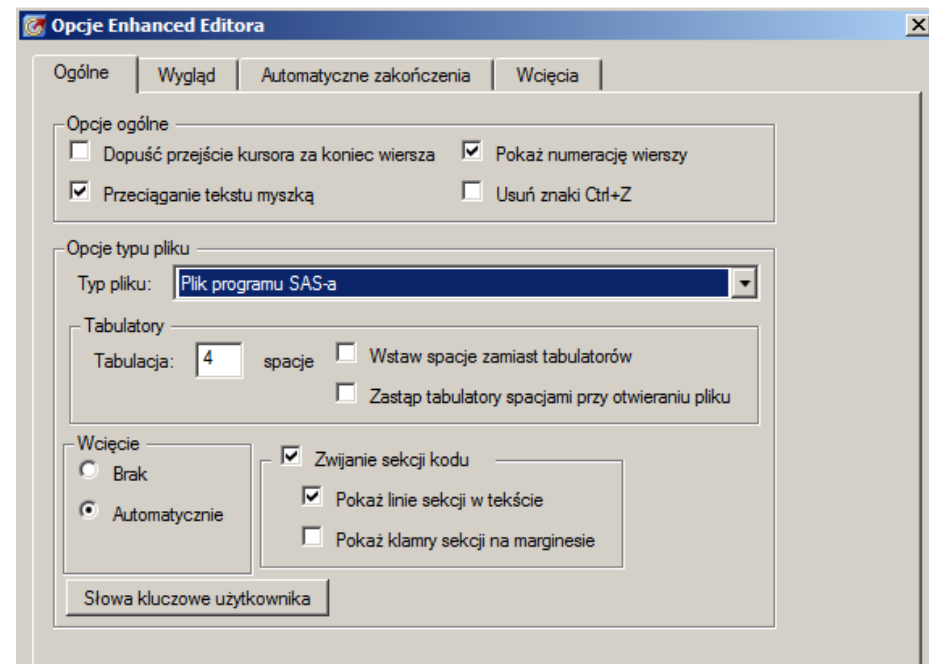
# Enhanced Editor – opcje

Opcje edytora są dostępne z menu głównego  
**Program->Opcje edytora.**



# Enhanced Editor – opcje

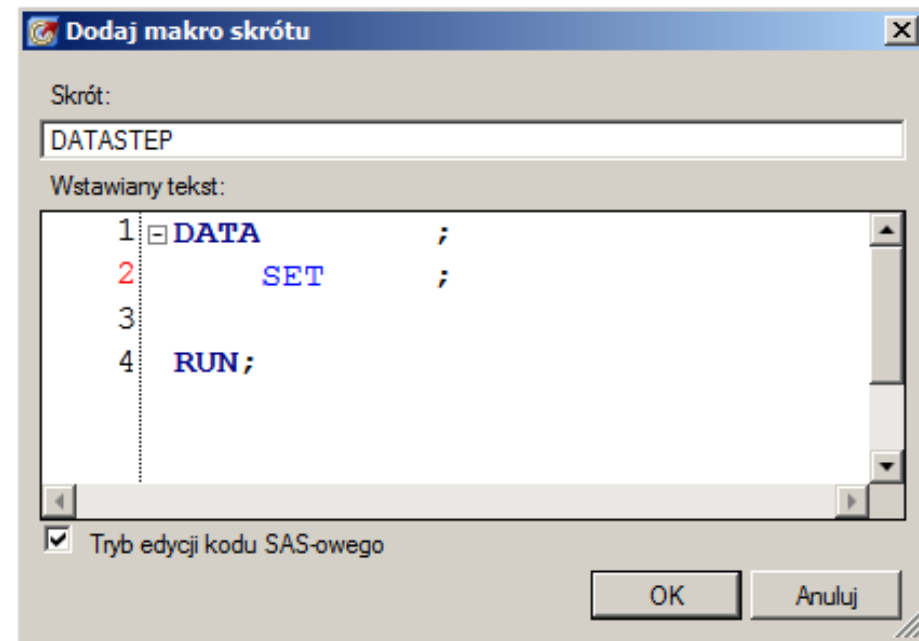
Opcje edytora są dostępne z menu głównego  
**Program->Opcje edytora.**



# Enhanced Editor – skróty

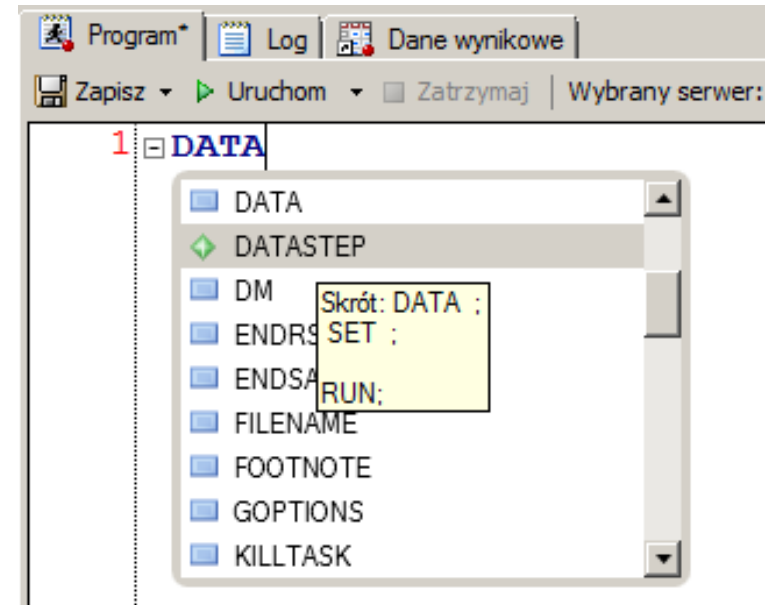
Enhanced Editor pozwala stworzyć skróty ułatwiające tworzenie programów. Jako skrót można zapisać najczęściej używane fragmenty kodu.

Skróty dostępne są z menu głównego **Program->Dodaj makro skrótu**

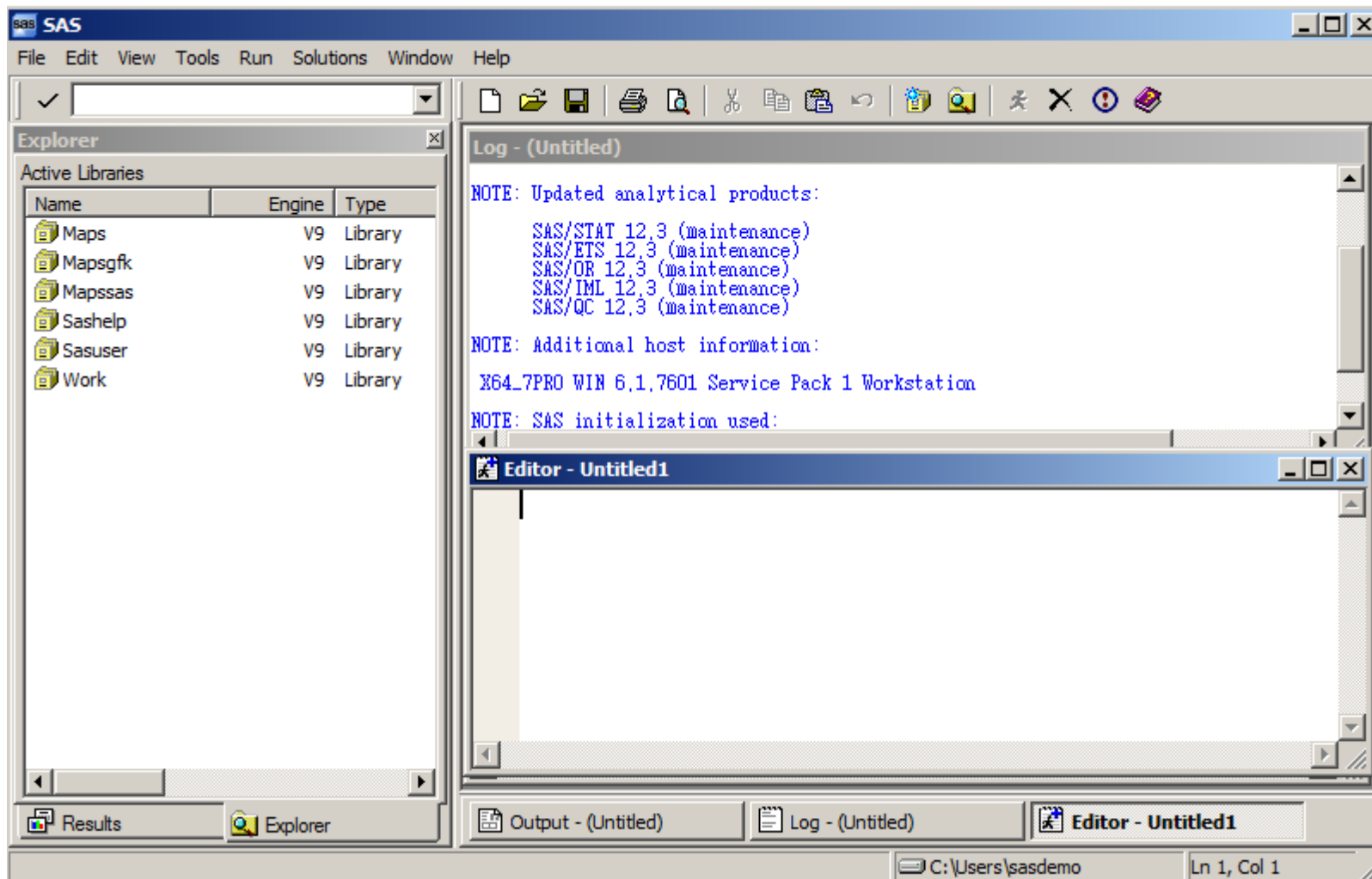


# Enhanced Editor – skróty

W celu użycia skrótu w edytorze należy napisać nazwę skrótu i nacisnąć Enter. W miejscu kursora skrót zostanie rozwinięty do tekstu umieszczonego w skrócie.



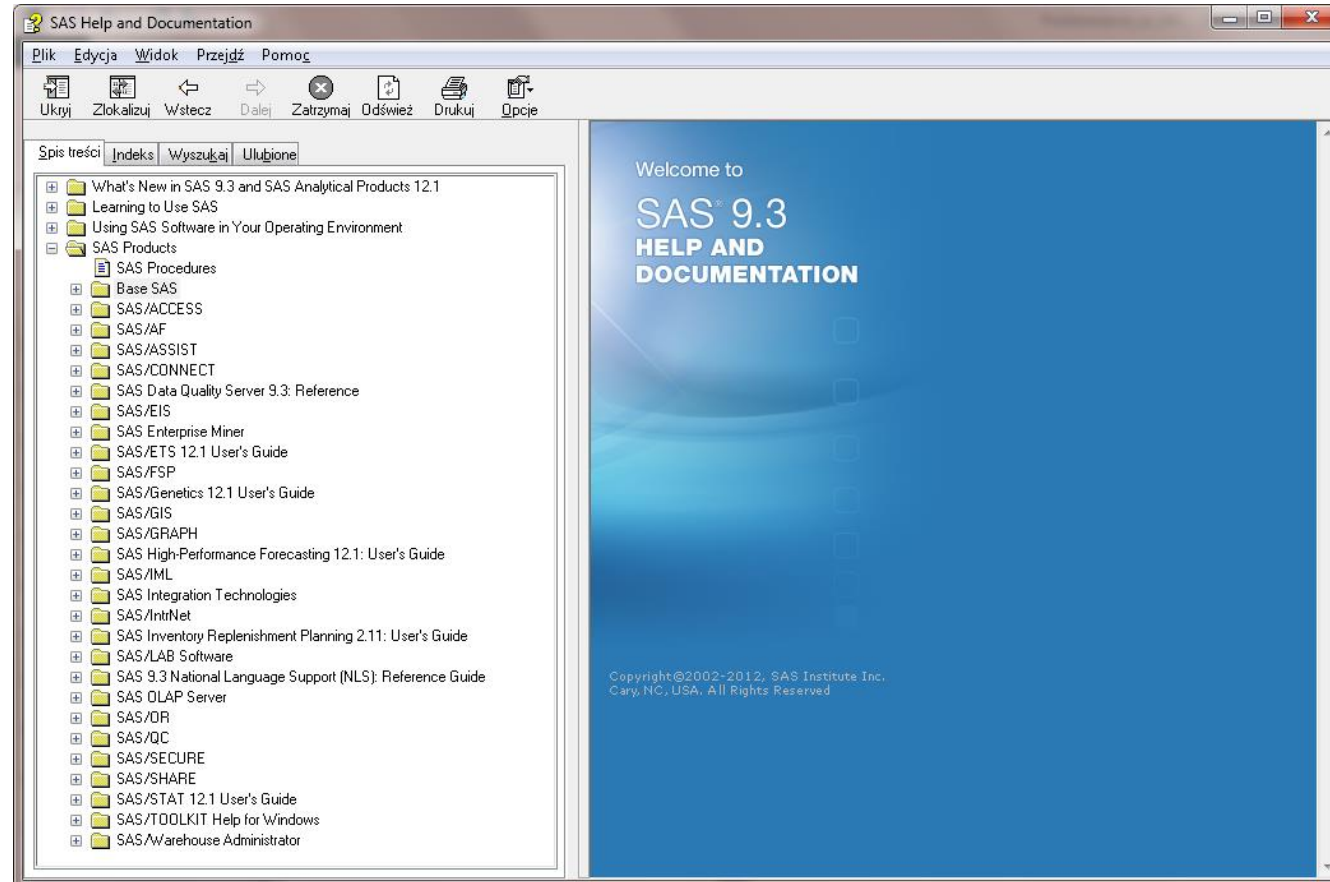
# SAS Display Manager – środowisko alternatywne



# Dokumentacja

- <http://support.sas.com/documentation/>

- 



# Sposób zapisu i przechowywania danych

- Format SAS: \*.sas7bdat
- Tabele innych baz danych
- Arkusze kalkulacyjne Excel
- Pliki tekstowe
- Pliki XML
- ...

# Biblioteka

- Repozytorium danych
- Logiczna struktura wskazująca miejsce przechowywania danych (referencja, wskaźnik).
- Definiuje sposób dostępu:
  - Typ danych
  - Lokalizację danych
  - Prawa dostępu



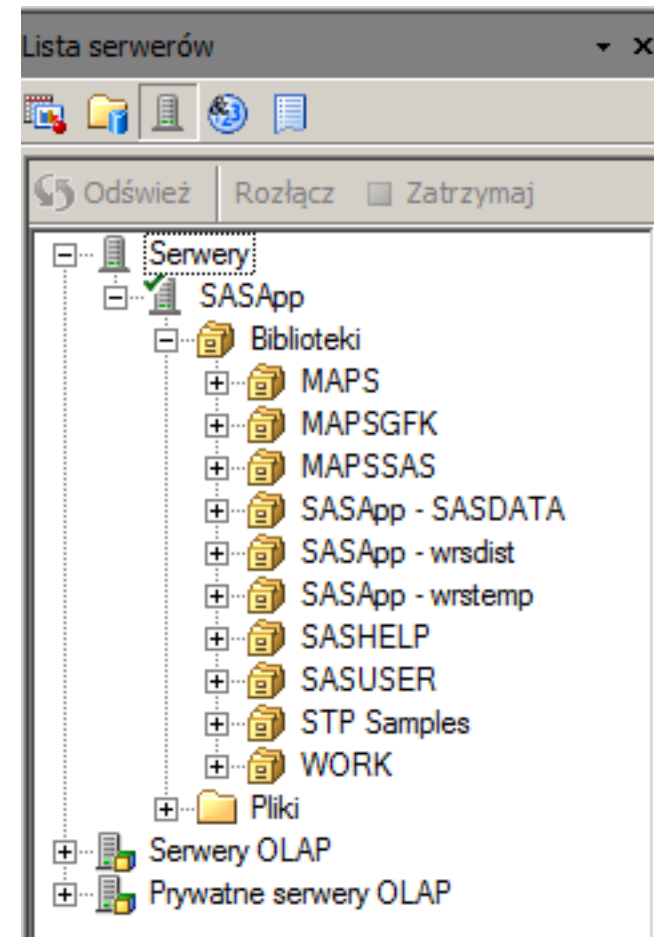
# Podstawowe biblioteki

- **WORK** – tymczasowa, domyślna, unikalna dla każdej sesji
- **SASUSER** – trwała, unikalna dla każdego użytkownika
- **MAPS, MAPSSAS, MAPSGFK** – zawiera dane pozwalające narysować mapę

# Biblioteki w SAS Enterprise Guide

Lista dostępnych bibliotek jest widoczna na **Lista serwerów**.

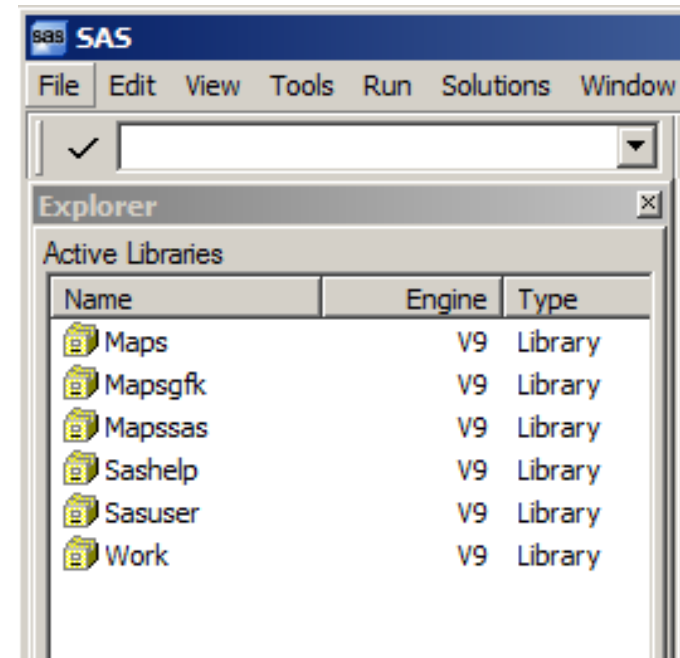
Biblioteki podpinane z kodu będą widoczne po kliknięciu przycisku **Odśwież**



# Biblioteki w SAS Display Manager

Lista dostępnych bibliotek jest widoczna w oknie Explorer po lewej stronie.

Biblioteki podpinane z kodu będą widoczne po automatycznie



# Biblioteki własne

- LIBNAME *libref* engine opcje;
  - LIBNAME bib base "c:\katalog";
  - LIBNAME bibxls excel "c:\plik.xls";\*
  - LIBNAME bibxls pcfiles path="c:\plik.xls";\*
- LIBNAME *libref* LIST;
- LIBNAME *libref* CLEAR;

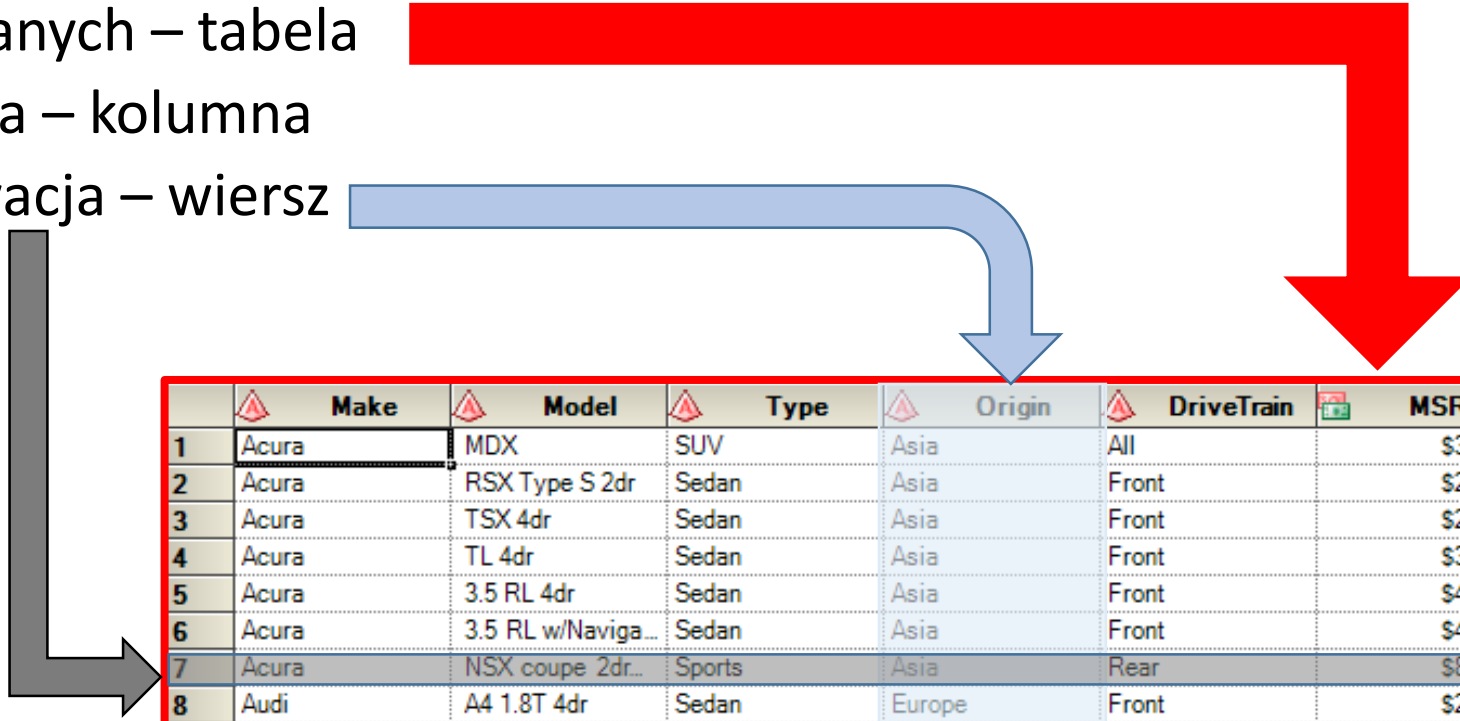
# Biblioteki – zaawansowane użycie

- Biblioteka do wielu folderów:
  - `libname kurs ("c:\data" "c:\sas");`
- Konkatenacje bibliotek:
  - `libname k1 "c:\data";`
  - `libname k2 "c:\sas";`
  - `libname kurs2 (k1 k2);`

# Zbiory danych

- Stosowana terminologia:

- Zbiór danych – tabela
- Zmienna – kolumna
- Obserwacja – wiersz



	Make	Model	Type	Origin	DriveTrain	MSRP
1	Acura	MDX	SUV	Asia	All	\$36,945
2	Acura	RSX Type S 2dr	Sedan	Asia	Front	\$23,820
3	Acura	TSX 4dr	Sedan	Asia	Front	\$26,990
4	Acura	TL 4dr	Sedan	Asia	Front	\$33,195
5	Acura	3.5 RL 4dr	Sedan	Asia	Front	\$43,755
6	Acura	3.5 RL w/Naviga...	Sedan	Asia	Front	\$46,100
7	Acura	NSX coupe 2dr...	Sports	Asia	Rear	\$89,765
8	Audi	A4 1.8T 4dr	Sedan	Europe	Front	\$25,940
9	Audi	A4 1.8T converti...	Sedan	Europe	Front	\$35,940
10	Audi	A4 3.0 4dr	Sedan	Europe	Front	\$31,840
11	Audi	A4 3.0 Quattro 4...	Sedan	Europe	All	\$33,430
12	Audi	A4 3.0 Quattro 4...	Sedan	Europe	All	\$34,480
13	Audi	A6 3.0 4dr	Sedan	Europe	Front	\$36,640
14	Audi	A6 3.0 Quattro 4...	Sedan	Europe	All	\$39,640

# Atrybuty zmiennych

- Nazwa: do 32 znaków
- Typ: numeryczny, znakowy
- Długość
- Format
- Informat
- Etykieta: do 256 znaków
- Typ indeksu: brak, prosty, złożony, oba
- Dodatkowe atrybuty

# Typ danych – numeryczny

- Stała numeryczna:
  - 10
  - 12.45
  - -5.18
  - 10e5
  - .
- Zmienna numeryczna
  - Długość 3-8 bajtów\*
  - Przechowuje wartości całkowitoliczbowe i zmiennoprzecinkowe



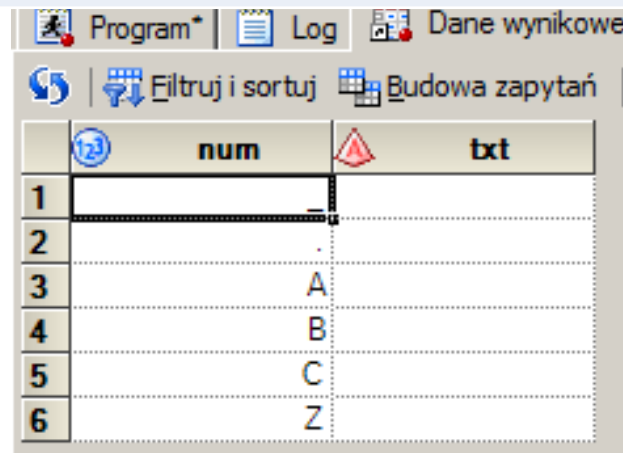
# Typ danych - znakowy

- Stała znakowa
  - 'tekst 1'
  - "Tekst 2"
  - ""
- Zmienna znakowa
  - Długość 1-32,767 bajtów
  - Czuła na wielkość liter

# Braki danych

Braki danych mogą być reprezentowane przez następujące symbole:

BRAK DANYCH	SYMBOL
Numeryczny	. (pojedyncza kropka)
Znakowa	' ' (spacja umieszczona w cudzysłowie lub apostrofach )
Specjalny	.litera (pojedyncza kropka z literą od A do Z)
Specjalny	._ (pojedyncza kropka z podkreśleniem)



The screenshot shows a software interface with a table. The window title bar includes 'Program\*', 'Log', and 'Dane wynikowe'. Below the title bar is a toolbar with icons for 'Filtruj i sortuj' and 'Budowa zapytań'. The table has two columns: 'num' and 'txt'. The 'num' column contains values 1 through 6. The 'txt' column contains a hyphen '-' for row 1, a period '.' for row 2, and letters 'A', 'B', 'C', and 'Z' for rows 3 through 6 respectively. A red warning triangle icon is positioned above the 'txt' column header.

	num	txt
1		-
2		.
3		A
4		B
5		C
6		Z

# Operatory

Dodawanie	+
Odejmowanie	-
Mnożenie	*
Dzielenie	/
Potęgowanie	**

Konkatenacja tekstów	
-------------------------	--

# Data oraz czas

- Data: wartość numeryczna jako liczba dni, która upłynęła od 1 stycznia 1960 roku (wartość 0)
  - Stała daty: "DDMONYYYY"d
- Czas: wartość numeryczna jako liczba sekund, która upłynęła do godziny 00:00 (wartość 0)
  - Stała czasu: "HH:mm:ss"t
- Data i czas: wartość numeryczna jako liczba sekund, która upłynęła od godziny 00:00 1 stycznia 1960 roku (wartość 0)
  - Stała daty i czasu "DDMONYYYY:HH:mm:ss"dt

# Stałe daty i czasu

```
data data_i_czas;  
    D   = "01JAN2015"D;  
    T   = "14:17:21"T;  
    DT  = "01JAN2015:14:17:21"DT;  
run;
```

- Minimalna data to:
  - **01JAN1582**
- Maksymalna stała daty to:
  - **"31DEC20000"d**

# Wprowadzenie do języka SAS 4GL

- Bloki instrukcji

- data step

- DATA *zbior\_wynikowy*;**

- ...

- RUN;**

- proc step

- PROC NAZWA\_PROCEDURY;**

- ...

- RUN; / QUIT;**

- Instrukcje wolne

Data step

# Data step składnia

```
DATA zb1 zb2 zb3;
```

```
...
```

```
RUN <CANCEL>;
```



# PDV – Program Data Vector

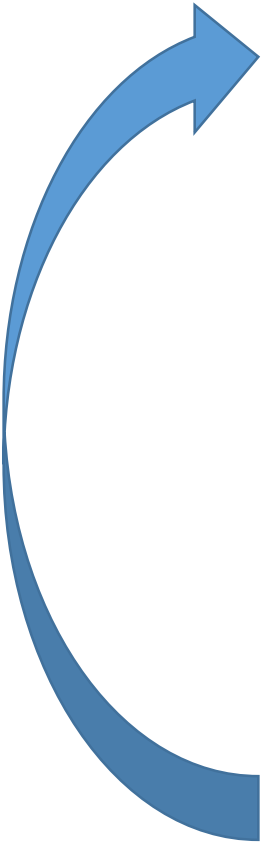
- Struktura przechowująca pojedynczą obserwację w trakcie działania data stepu
  - `_N_`: zmienna automatyczna przechowująca numer iteracji data stepu
  - `_ERROR_`: zmienna automatyczna przechowująca numer iteracji
  - `num_var`, `char_var`: przykładowe zmienne użytkownika

<code>_N_</code>	<code>_ERROR_</code>	<code>num_var</code>	<code>char_var</code>
NUMERIC	NUMERIC	NUMERIC	CHARACTER
1	0	.	

# Instrukcja

- SET – wczytuje obserwacje sekwencyjnie ze wskazanego zbioru/zbiorów danych.
- INFILE – wczytuje wiersze sekwencyjnie ze wskazanego pliku/plików.
- OUTPUT – zapisuje wektor PDV do zbioru/zbiorów wynikowych
- RETURN – wykonuje skok wykonania pętli datastepu na początek.

# Data step – zasada działania



```
DATA zb_out;  
  SET zb_in;  
  instrukcja1;  
  instrukcja2;
```

```
  output;  
  return;
```

```
RUN;
```

zb\_in

ZMIENNA_A	ZMIENNA_B
1	A
2	B
3	C
4	D

**Instrukcje wykonywane automatycznie.  
Nie ma konieczności umieszczanie ich w  
datastepie.**

# Data step – zasada działania

```
DATA zb_out;
```

```
SET zb_in;
```

```
USR = zmienna_a * 2;
```

```
output;
```

```
return;
```

```
RUN;
```

zb\_in

ZMIENNA_A	ZMIENNA_B
1	A

PDV

_N_	_ERROR_	ZBZMIENNA_A	ZMIENNA_B	USR
1	0	1	A	.

ZMIENNA_A	ZMIENNA_B	USR
-----------	-----------	-----

zb\_out

# Data step – zasada działania

```
DATA zb_out;
```

```
  SET zb_in;
```

```
  USR = zmienna_a * 2;
```

```
output;
```

```
return;
```

```
RUN;
```

zb\_in

ZMIENNA_A	ZMIENNA_B
1	A

PDV

_N_	_ERROR_	ZBZMIENNA_A	ZMIENNA_B	USR
1	0	1	A	2

ZMIENNA_A	ZMIENNA_B	USR
-----------	-----------	-----

zb\_out

# Data step – zasada działania

```
DATA zb_out;  
  SET zb_in;  
  USR = zmienna_a * 2;
```

*output;*  
*return;*

```
RUN;
```

zb\_in

ZMIENNA_A	ZMIENNA_B
1	A

PDV

_N_	_ERROR_	ZBZMIENNA_A	ZMIENNA_B	USR
1	0	1	A	2

ZMIENNA_A	ZMIENNA_B	USR
1	A	2

zb\_out

# Data step – zasada działania

DATA zb\_out;  
SET zb\_in;  
USR = zmienna\_a \* 2;

*output;*  
*return;*

RUN;

zb\_in

ZMIENNA_A	ZMIENNA_B
1	A

PDV

_N_	_ERROR_	ZBZMIENNA_A	ZMIENNA_B	USR
1	0	1	A	2

ZMIENNA_A	ZMIENNA_B	USR
1	A	2

zb\_out

# Data step – zasada działania

```
DATA zb_out;
```

```
SET zb_in;
```

```
USR = zmienna_a * 2;
```

```
output;
```

```
return;
```

```
RUN;
```

zb\_in

ZMIENNA_A	ZMIENNA_B
2	B

PDV

_N_	_ERROR_	ZBZMIENNA_A	ZMIENNA_B	USR
2	0	2	B	.

ZMIENNA_A	ZMIENNA_B	USR
1	A	2

zb\_out



# Data step – zasada działania

```
DATA zb_out;
```

```
  SET zb_in;
```

```
  USR = zmienna_a * 2;
```

```
output;
```

```
return;
```

```
RUN;
```

zb\_in

ZMIENNA_A	ZMIENNA_B
2	B

PDV

_N_	_ERROR_	ZBZMIENNA_A	ZMIENNA_B	USR
2	0	2	B	4

ZMIENNA_A	ZMIENNA_B	USR
1	A	2

zb\_out

# Data step – zasada działania

```
DATA zb_out;  
  SET zb_in;  
  USR = zmienna_a * 2;
```

*output;*  
*return;*

```
RUN;
```

zb\_in

ZMIENNA_A	ZMIENNA_B
2	B

PDV

_N_	_ERROR_	ZBZMIENNA_A	ZMIENNA_B	USR
2	0	2	B	4

zb\_out

ZMIENNA_A	ZMIENNA_B	USR
1	A	2
2	B	4

# Data step – zasada działania



```
DATA zb_out;  
  SET zb_in;  
  USR = zmienna_a * 2;
```

*output;*  
*return;*

RUN;

zb\_in

ZMIENNA_A	ZMIENNA_B
2	B

PDV

_N_	_ERROR_	ZBZMIENNA_A	ZMIENNA_B	USR
2	0	2	B	4

zb\_out

ZMIENNA_A	ZMIENNA_B	USR
1	A	2
2	B	4

# Data step – zasada działania

```
DATA zb_out;
```

```
SET zb_in;
```

```
USR = zmienna_a * 2;
```

```
output;
```

```
return;
```

```
RUN;
```

zb\_in

ZMIENNA_A	ZMIENNA_B
3	C

PDV

_N_	_ERROR_	ZBZMIENNA_A	ZMIENNA_B	USR
3	0	3	C	.

zb\_out

ZMIENNA_A	ZMIENNA_B	USR
1	A	2
2	B	4

# Data step – zasada działania

```
DATA zb_out;
```

```
  SET zb_in;
```

```
  USR = zmienna_a * 2;
```

```
output;
```

```
return;
```

```
RUN;
```

zb\_in

ZMIENNA_A	ZMIENNA_B
3	C

PDV

_N_	_ERROR_	ZBZMIENNA_A	ZMIENNA_B	USR
3	0	3	C	6

zb\_out

ZMIENNA_A	ZMIENNA_B	USR
1	A	2
2	B	4

# Data step – zasada działania

```
DATA zb_out;  
  SET zb_in;  
  USR = zmienna_a * 2;
```

*output;*  
*return;*

```
RUN;
```

zb\_in

ZMIENNA_A	ZMIENNA_B
3	C

PDV

_N_	_ERROR_	ZBZMIENNA_A	ZMIENNA_B	USR
3	0	3	C	6

zb\_out

ZMIENNA_A	ZMIENNA_B	USR
1	A	2
2	B	4
3	C	6

# Data step – zasada działania

DATA zb\_out;  
SET zb\_in;  
USR = zmienna\_a \* 2;

*output;*  
*return;*

RUN;

zb\_in

ZMIENNA_A	ZMIENNA_B
3	C

PDV

_N_	_ERROR_	ZBZMIENNA_A	ZMIENNA_B	USR
3	0	3	C	6

zb\_out

ZMIENNA_A	ZMIENNA_B	USR
1	A	2
2	B	4
3	C	6

# Data step – zasada działania

```
DATA zb_out;  
  SET zb_in;  
  USR = zmienna_a * 2;
```

```
output;  
return;
```

```
RUN;
```

zb\_in

ZMIENNA_A	ZMIENNA_B
4	D

PDV

_N_	_ERROR_	ZBZMIENNA_A	ZMIENNA_B	USR
4	0	4	D	.

zb\_out

ZMIENNA_A	ZMIENNA_B	USR
1	A	2
2	B	4
3	C	6



# Data step – zasada działania

```
DATA zb_out;
```

```
  SET zb_in;
```

```
  USR = zmienna_a * 2;
```

```
output;
```

```
return;
```

```
RUN;
```

zb\_in

ZMIENNA_A	ZMIENNA_B
4	D

PDV

_N_	_ERROR_	ZBZMIENNA_A	ZMIENNA_B	USR
4	0	4	D	8

zb\_out

ZMIENNA_A	ZMIENNA_B	USR
1	A	2
2	B	4
3	C	6

# Data step – zasada działania

```
DATA zb_out;  
  SET zb_in;  
  USR = zmienna_a * 2;  
  
output;  
return;  
  
RUN;
```

zb\_in

ZMIENNA_A	ZMIENNA_B
4	D

PDV

_N_	_ERROR_	ZBZMIENNA_A	ZMIENNA_B	USR
4	0	4	D	8

zb\_out

	ZMIENNA_A	ZMIENNA_B	USR
1	A		2
2	B		4
3	C		6
4	D		8

# Data step – zasada działania

```
DATA zb_out;  
  SET zb_in;  
  USR = zmienna_a * 2;
```



*output;*  
*return;*

RUN;

zb\_in

ZMIENNA_A	ZMIENNA_B
4	D

PDV

_N_	_ERROR_	ZBZMIENNA_A	ZMIENNA_B	USR
3	0	3	C	6

zb\_out

ZMIENNA_A	ZMIENNA_B	USR
1	A	2
2	B	4
3	C	6
4	D	8

# Data step – zasada działania

DATA zb\_out;

SET zb\_in;

USR = zmienna\_a \* 2;



zb\_in

ZMIENNA_A	ZMIENNA_B

PDV

_N_	_ERROR_	ZBZMIENNA_A	ZMIENNA_B	USR
4	0	4	D	.

*output;*

*return;*

RUN;

zb\_out

	ZMIENNA_A	ZMIENNA_B	USR
1	A		2
2	B		4
3	C		6
4	D		8

# Zakresy zmiennych

Odwołania do zmiennych można wykonać na następujące sposoby:

- Bezpośrednie odwołanie:
  - var1 var2 var3
- Zakres zmiennych zawierających stały wzorzec i indeksowany sufix:
  - var1-var3
- Zakres zmiennych z wektora PDV (wszystkie zmienne z wektora pomiędzy pierwszą i ostatnią z zakresu):
  - a – z
- Zmienne zaczynające się od:
  - var1:
- Wszystkie zmienne ze zbioru lub wszystkie zmienne danego typu:
  - \_ALL\_    \_NUMERIC\_    \_CHARACTER\_

# Data step – tworzenie zmiennych

Zmienną w obrębie data step można stworzyć na wiele sposobów.  
Najpopularniejsze to:

- Instrukcja LENGTH
- Przypisanie wartości do zmiennej;

# Data step – tworzenie zmiennych

Instrukcja LENGTH:

***LENGTH \_variable\_list\_ <\$>n;***

- **\_variable\_list\_**: lista zmiennych, którym chcemy przypisać tą samą długość.
- **n** : długość zmiennych w bajtach mieszcząca się w granicach dla odpowiedniego typu zmiennej.
- **\$** : długość zmiennej znakowej musi być poprzedzona znakiem \$.

# Data step – tworzenie zmiennych

Instrukcja LENGTH:

***DATA zmienne;***

***Length txt \$20;***

***length n1 n2 n3 8;***

***RUN;***



# Data step – tworzenie zmiennych

Przypisanie wartości do zmiennej:

***\_var\_name = constant / expression;***

- **Constant:** stała numeryczna lub stała tekstowa.
- **Expression :** wyrażenie wykorzystujące inne zmienne lub funkcje.





# Data step – tworzenie zmiennych

```
1 DATA nowe_zmienne;  
2     length txt1 txt2 $12;  
3     length num1 num2 num3 8 txt3 $8;  
4     num4 = 123;  
5     txt4 = "Piotr Rozenbajgier";  
6 RUN;
```

# Data step – tworzenie zmiennych

```
1 DATA nowe_zmienne;  
2     length txt1 txt2 $12;  
3     length num1 num2 num3 8 txt3 $8;  
4     num4 = 123;  
5     txt4 = "Piotr Rozenbajgier";  
6 RUN;
```

Właściwości NOWE_ZMIENNE		
Ogólne	Kolumny	
Kolumny		
Zaawansowane		
Podsumowanie		
Nazwa	Typ	Długość
bt1	znakowa	12
bt2	znakowa	12
num1	numeryczna	8
num2	numeryczna	8
num3	numeryczna	8
bt3	znakowa	8
num4	numeryczna	8
bt4	znakowa	18

	 txt1	 txt2	 num1	 num2	 num3	 txt3	 num4	 txt4
1			.	.	.		123	Piotr Rozenbajgier

# WHERE – ograniczanie wiersze

W celu ograniczenia wierszy przetwarzanych w data stepie lub procedurze należy wykorzystać WHERE:

- Opcja zbioru
  - WHERE= (warunek logiczny)
- Instrukcja
  - WHERE warunek logiczny

# WHERE – przykład

- Opcje zbioru:

```
DATA boys(where=(sex='M')) girls(where=(sex='F'));  
      SET sashelp.class(where=(age=14));  
RUN;
```

- Instrukcja:

```
DATA boys14  
      SET sashelp.class;  
      where age=14 and sex='M';  
RUN;
```

# KEEP/DROP – ogranicza kolumny

W celu ograniczenia kolumn przewarzanych w data stepie należy wykorzystać KEEP lub DROP:

- Opcje zbioru:
  - KEEP = lista zmiennych
  - DROP = lista zmiennych
- Instrukcja
  - KEEP lista zmiennych
  - DROP lista zmiennych

# KEEP/DROP – przykład

- Opcje zbioru:

```
DATA class(drop=height);  
    SET sashelp.class(keep=name age weight height);  
RUN;
```

- Instrukcja:

```
DATA class  
    SET sashelp.class;  
    drop height sex;  
RUN;
```

# RENAME – zmiana nazwy zmiennej

Aby zmienić nazwę kolumny podczas przetwarzania data stepie lub procedurze należy użyć RENAME:

- Opcje zbioru:
  - RENAME = (stara nazwa = nowa nazwa)
- Instrukcja
  - RENAME stara nazwa = nowa nazwa



# RENAME – przykład

- Opcje zbioru:

```
DATA class(rename=(height=wzrost));  
    SET sashelp.class(rename=(weight=waga));  
RUN;
```

- Instrukcja:

```
DATA class  
    SET sashelp.class  
    rename height=wzrost weight=waga;  
RUN;
```

# FIRSTOBS / OBS – ogranicza wiersze

W celu ograniczenia liczby przetwarzanych wierszy w data stepie i procedurach należy wykorzystać opcje zbioru FIRSTOBS i/lub OBS:

- Numer pierwszej czytanej obserwacji
  - FIRSTOBS = stała numeryczna
- Numer ostatniej czytanej obserwacji
  - OBS = stała numeryczna

# FIRSTOBS / OBS –przykład

**DATA class**

**SET sashelp.class(firstobs=2 obs=15);**

**RUN;**

# Funkcje


# Data i czas

- **DATE()/TODAY()** – zwraca bieżącą datę
- **DATETIME()** – zwraca bieżącą datę i czas
- **TIME()** – zwraca bieżący czas
- **DATEPART(data\_czas)** – zwraca część datę z podanej daty i czasu
- **TIMEPART(data\_czas)** – zwraca część czasu z podanej daty i czasu

# Data i czas

Instrukcja format będzie omówiona w dalszej części kursu.


```
1 data data_i_czas;  
2     d = DATE();  
3     dt = DATETIME();  
4     t = TIME();  
5     format d date9.  
6           dt datetime18.  
7           t time9.;  
8 run;
```



	d	dt	t
1	03SEP2015	03SEP15:20:45:35	20:45:35

# Data i czas

```
1 data data_i_czas;  
2     dt = DATETIME();  
3     dpart = DATEPART(dt);  
4     tpart = TIMEPART(dt);  
5     format dpart date9.  
6           dt datetime18.  
7           tpart time9.;  
8 run;
```



Program*			Log	Dane wynikowe
Filtruj i sortuj			Budowa zapytań	Dane ▾ Opisowe
	dt	dpart	tpart	
1	03SEP15:20:53:30	03SEP2015	20:53:30	


# Data i czas

- **DAY(data)** – zwraca numer dnia w miesiącu z podanej daty
- **WEEK(data)** – zwraca numer tygodnia w roku z podanej daty
- **WEEKDAY(data)** – zwraca numer dnia tygodnia z podanej daty
- **MONTH(data)** – zwraca numer miesiąca w roku z podanej daty
- **QTR(data)** – zwraca numer kwartału w roku z podanej daty
- **YEAR(data)** – zwraca numer roku z podanej daty



# Data i czas

```
1 data data_i_czas;  
2     d = date();  
3     year = year(d);  
4     quarter = qtr(d);  
5     month=month(d);  
6     day=day(d);  
7     week=week(d);  
8     weekday=weekday(d);  
9  
10    format d date9.;  
11    run;
```




Program*   Log   Dane wynikowe							
Filtruj i sortuj   Budowa zapytań   Dane ▾   Opisowe ▾   Wykres ▾   Analizuj ▾   Eks							
	d	year	quarter	month	day	week	weekday
1	03SEP2015	2015	3	9	3	35	5

# Data i czas

- **MDY(mm,dd,yyyy)** – zwraca datę z podanego miesiąca, dnia i roku
- **DHMS(data, hh, ms, ss)** – zwraca datę i czas z podanej daty, godziny, minuty i sekundy
- **HMS(hh, ms, ss)** – zwraca czas z czas z podanej godziny, minuty i sekundy

# Data i czas

```
1 data data_i_czas;  
2     d = mdy(9,3,2015);  
3     t = hms(20,15,13);  
4     dt= dhms(d,20,15,13);  
5  
6     format d date9.  
7           t time9.  
8           dt time18.;  
9 run;
```




	d	t	dt
1	03SEP2015	20:15:13	488036:15:13

# Data i czas

- **hour(czas)** – zwraca godzinę z podanego czasu
- **minute(czas)** – zwraca minutę z podanego czasu
- **second(czas)** – zwraca sekundę z podanego czasu

# Data i czas

```
1 data data_i_czas;  
2     t = time();  
3     h = hour(t);  
4     m = minute(t);  
5     s = second(t);  
6     format t time9.;  
7 run;
```




	t	h	m	s
1	21:05:28	21	5	28.440999985

# Data i czas

- **INTNX(okres, start, N, wyrównanie)** – zwraca datę start przesuniętą o N okresów i wyrównaną do początku/końca/środku okresu. Np. ostatni dzień następnego miesiąca.
- **INTCK(okres, start, koniec)** – zwraca liczbę granic okresu pomiędzy datami start i koniec. Np. liczba lat pomiędzy datami.

# Data i czas

```
1 data data_i_czas;  
2     dzis=today();  
3     d = intnx('month', dzis, -5, "B");  
4     count = intck('month', d, dzis);  
5  
6     format dzis d date9.;  
7 run;
```



	Program*	Log	Dane wynikowe
	Filtruj i sortuj	Budowa zapytań	Dane ▾ Opisowe
	dzis	d	count
1	03SEP2015	01APR2015	5


# Zaokrąglenia

- **FLOOR(liczba)** – zwraca największą liczbę całkowitą mniejszą lub równą od podanego argumentu
- **CEIL(liczba)** – zwraca najmniejszą liczbę całkowitą większą lub równą od podanego argumentu
- **INT(liczba)** – zwraca część całkowitą z podanego argumentu
- **FUZZ(liczba)** – zwraca liczbę najbliższą liczbę całkowitą jeśli argument różni się od tej liczby o mniej niż 1E-12
- **ROUND(arg1, arg2)** – zwraca arg1 zaokrąglony do najbliższej wielokrotności arg2



# Zaokrąglenia

```
1 data zaokrąglenia;  
2     liczba = 128.14;  
3     f = FLOOR(liczba);  
4     c = CEIL(liczba);  
5     i = INT(liczba);  
6     fz = FUZZ(liczba);  
7     r = ROUND(liczba , 10);  
8 run;
```




	liczba	f	c	i	fz	r
1	128.14	128	129	128	128.14	130

# Liczby losowe

- **RAND(dist, par1,...par-k)** – zwraca liczbę losową z podanego rozkładu
- **RANEXP(seed)** – zwraca liczbę losową z rozkładu wykładniczego
- **RANNOR(seed)/NORMAL(seed)** – zwraca liczbę z rozkładu normalnego
- **RANUNI(seed)/UNIFORM(seed)** – zwraca liczbę z rozkładu jednostajnego

# Liczby losowe

```
1 data liczby_losowe;  
2     ranexp = RANEXP(12345);  
3     rannor = RANNOR(12345);  
4     ranuni = RANUNI(12345);  
5 run;
```




	ranexp	rannor	ranuni
1	1.0135605834	0.373945642	0.276277173

# Matematyczne

- **ABS(liczba)** – zwraca wartość absolutną z podanej liczby
- **EXP(liczba)** – zwraca liczbę będącą wynikiem działania  $e^{\text{liczba}}$
- **LOG(liczba)** – zwraca logarytm naturalny z podanej liczby
- **LOG10(liczba)** – zwraca logarytm dziesiętny z podanej liczby
- **MOD(arg1, arg2)** – zwraca resztę z dzielenia arg1 przez arg2
- **SQRT(liczba)** – zwraca pierwiastek kwadratowy z podanej liczby
- **SIGN(liczba)** – zwraca znak podanej liczby

# Matematyczne

```
1 data liczby_losowe;  
2     liczba = 12.56;  
3     abs = ABS(liczba);  
4     exp = EXP(liczba);  
5     log = LOG(liczba);  
6     log10 = LOG10(liczba);  
7     mod = MOD(liczba, 5);  
8     sgrt = SQRT(liczba);  
9     sign = SIGN(liczba);  
10 run;
```




	liczba	abs	exp	log	log10	mod	sgrt	sign
1	12.56	12.56	284930.33763	2.530517161	1.0989896394	2.56	3.5440090293	1

# Statystyka opisowa

- **MIN(arg1 <,arg2...>)** – zwraca minimum z podanych argumentów
- **MAX(arg1 <,arg2...>)** – zwraca maksimum z podanych argumentów
- **MEAN(arg1 <,arg2...>)** – zwraca średnią z podanych argumentów
- **MEDIAN(arg1 <,arg2...>)** – zwraca medianę z podanych argumentów
- **SUM(arg1 <,arg2...>)** – zwraca sumę z podanych argumentów

# Statystyka opisowa


```
1 data matematyczne;  
2     l1=15;  
3     l2=13;  
4     min = min(l1, l2);  
5     max = max(l1, l2);  
6     mean = mean(l1, l2);  
7     median = median(l1, l2);  
8     sum = sum(l1, l2);  
9 run;
```



	l1	l2	min	max	mean	median	sum
1	15	13	13	15	14	14	28

# Statystyka opisowa

```
1 data matematyczne;  
2     l1=15;  
3     l2=13;  
4     min = min(of l1 - l2);  
5     max = max(of l1 - l2);  
6     mean = mean(of l1 - l2);  
7     median = median(of l1 - l2);  
8     sum = sum(of l1 - l2);  
9 run;
```



	l1	l2	min	max	mean	median	sum
1	15	13	13	15	14	14	28



# Statystyka opisowa

- **N(arg1 <,arg2...>)** – zwraca liczbę argumentów niebędących brakiem danych
- **NMISS(arg1 <,arg2...>)** – zwraca liczbę argumentów będących brakiem danych
- **RANGE(arg1 <,arg2...>)** – zwraca zakres wartości z podanych argumentów
- **STD(arg1, arg2<,arg3...>)** – zwraca odchylenie standardowe z podanych argumentów
- **VAR(arg1, arg2<,arg3...>)** – zwraca wariancję z podanych argumentów

# Statystyka opisowa

```
1 data matematyczne;  
2     l1=15;  
3     l2=13;  
4     n = n(of l1 - l2);  
5     nmiss = nmiss(of l1 - l2);  
6     range = range(of l1 - l2);  
7     std = std(of l1 - l2);  
8     var = var(of l1 - l2);  
9 run;
```




Program*   Log   Dane wynikowe							
Filtruj i sortuj   Budowa zapytań   Dane ▾ Opisowe ▾ Wykres ▾ Analizuj ▾							
	l1	l2	n	nmiss	range	std	var
1	15	13	2	0	2	1.4142135624	2

# Znakowe

- **LOWCASE(tekst)** – zwraca tekst przekształcony na małe litery
- **UPCASE(tekst)** – zwraca tekst przekształcony na wielkie litery
- **LENGTH(tekst)** – zwraca długość podanego tekstu
- **CAT(txt1, txt2, ...)** – zwraca tekst będący połączeniem wszystkich argumentów
- **FIND(str, substr)** – zwraca pozycję wystąpienia substr w str.

# Znakowe

```
1 data znakowe;  
2     t1= " Przykładowy Tekst";  
3     t2="Tekst";  
4     lowercase=lowercase(t2);  
5     upcase=upcase(t2);  
6     cat = cat(t1, t2);  
7     find=find(t1, t2);  
8 run;
```



	t1	t2	lowercase	upcase	cat	find
1	Przykładowy Tekst	Tekst	tekst	TEKST	Przykładowy TekstTekst	14

# Znakowe

- **TRIM(tekst)** – zwraca tekst z usuniętymi spacjami z prawej strony tekstu
- **STRIP(tekst)** – zwraca tekst z usuniętymi spacjami z lewej i prawej strony tekstu
- **LEFT(tekst)** – zwraca tekst przesunięty do lewej strony
- **REVERSE(tekst)** – zwraca tekst pisany wspak
- **QUOTE(tekst)** – zwraca tekst umieszczony w cudzysłów.

# Znakowe

```
1 data znakowe;  
2     t1 = "      tekst ";  
3     trim = trim(t1);  
4     strip = strip(t1);  
5     left = left(t1);  
6     reverse = reverse(t1);  
7     quote = quote(t1);  
8 run;
```




	t1	trim	strip	left	reverse	quote
1	tekst	tekst	tekst	tekst	tsket	" tekst "

# Znakowe

- **SUBSTR(tekst, pos, L)** – zwraca fragment tekstu o długości L rozpoczynając od znaku pos
- **SCAN(txt, n <,del>)** – zwraca wyraz o numerze n z txt. Wyrazy oddzielane są separatorem del
- **COALESCEC(txt1, txt2, ...)** – zwraca pierwszy tekst niebędący brakiem danych

# Znakowe

```
1 data znakowe;  
2     t1 = "Przykładowy tekst ";  
3     t2 = ' ';  
4     substr = substr(t1, 5, 4);  
5     scan = scan(t1, 2, ' ');  
6     coalescec= coalescec(t2, t1);  
7 run;
```



	t1	t2	substr	scan	coalescec
1	Przykładowy tekst		klad	tekst	Przykładowy tekst



Tablice – array

# Tablica

- Tablica to tymczasowa struktura grupująca zmienne.
- Tablica zawsze grupuje zmienne z wektora PDV.
- Tablica nie tworzy kopii/duplikatów istniejących zmiennych.
- Elementy tablicy mogą być tylko jednego typu.
- Tablica zawsze tworzona jest przez instrukcję ARRAY.

# Array

**ARRAY** array-name { subscript } <\$> <length> <array-elements> <(initial-value-list)> ;

- array-name: poprawna nazwa SAS, inna niż dowolna zmienna z PDV
- subscript: opisuje rozmiar i indeksy tablicy:
  - N – tablica N elementowa o indeksach 1:N
  - N:M – tablica M-N elementowa o indeksach N:M
  - \* - rozmiar tablicy określony na podstawie **array-elements**
- \$: określa elementy tablicy jako znakowe
- length: określa długość każdego elementu tablicy / zmiennej.
- array-elements: określa nazwy elementów tablicy
- initial-value-list: określa początkowe wartości elementów tablicy

# Tablica – przykłady

- ARRAY simple {5};
- ARRAY complex {2,3};
- ARRAY x{5,3} score1-score15;
- ARRAY tab{\*} a b c;
- ARRAY num{\*} \_NUMERIC\_;
- ARRAY char{\*} \$ \_CHARACTER\_;
- ARRAY tab{\*} \_ALL\_;
- ARRAY simple{3} a b c (1 2 3);
- ARRAY simple{1:3} a b c (2\*5 10);

# Tablica tymczasowa – temporary array

- Tablica tymczasowa – tworzy grupę elementów danych.
- Rozmiar tablicy tymczasowej nie może być definiowany za pomocą \*.
- Elementy tablicy zachowują się jak zmienne w Data stepie z wyjątkami:
  - Nie mają nazwy – odwołanie do nich następuje jedynie poprzez tablicę,
  - Nie trafiają do PDV,
  - Nie trafiają do zbioru wynikowego.

# Tablica tymczasowa - przykłady

- ARRAY simple {5} \_TEMPORARY\_ (1 2 3 4 5);
- ARRAY complex {2,3} \_TEMPORARY\_;
- ARRAY x{5,3} \_TEMPORARY\_ (5\*1 5\*2 5\*3);
- ARRAY tab{10} \_TEMPORARY\_;

# Funkcje tablicowe

- **DIM<n>(nazwa-tablicy)** – zwraca rozmiar n-ego wymiaru podanej tablicy.
- **LBOUND<n>(nazwa-tablicy)** – zwraca minimalny indeks n-ego wymiaru podanej tablicy
- **HBOUND<n>(nazwa-tablicy)** – zwraca maksymalny indeks n-ego wymiaru podanej tablicy

# Użycie tablic

```
1 DATA tablice;  
2     ARRAY x{3} (1 2 3);  
3     dim = DIM(x);  
4     lower_bound = LBOUND(x);  
5     higher_bound = HBOUND(x);  
6     sum = SUM(of x(*));  
7     PUT x{*}=;  
8 RUN;
```



# Użycie tablic

```
1 DATA tablice;  
2     ARRAY x{3} (1 2 3);  
3     dim = DIM(x);  
4     lower_bound = LBOUND(x);  
5     higher_bound = HBOUND(x);  
6     sum = SUM(of x(*));  
7     PUT x{*}=;  
8 RUN;
```

	 x1	 x2	 x3	 dim	 lower_bound	 higher_bound	 sum
1	1	2	3	3	1	3	6

# Użycie tablic tymczasowych

```
1 DATA premia;  
2     ARRAY premia{3} _TEMPORARY_ (0.15 0.1 0.05);  
3     pracownik = 1;  
4     wartosc = premia(pracownik);  
5 RUN;
```

# Użycie tablic tymczasowych

```
1 DATA premia;  
2     ARRAY premia{3} _TEMPORARY_ (0.15 0.1 0.05);  
3     pracownik = 1;  
4     wartosc = premia(pracownik);  
5 RUN;
```

	 pracownik	 wartosc
1	1	0.15

Petle

# Pętle

Są trzy rodzaje pętli:

- **DO:** pętla wykonuje grupę instrukcji pomiędzy DO i END określoną (za pomocą zmiennej indeksującej) liczbę razy
- **DO WHILE:** pętla wykonuje grupę instrukcji pomiędzy DO i END tak długo jak warunek logiczny pozostaje prawdziwy. Warunek sprawdzany jest na początku pętli.
- **DO UNTIL:** pętla wykonuj grupę instrukcji pomiędzy DO i END tak długo jak warunek logiczny pozostaje fałszywy. Warunek sprawdzany jest na końcu pętli.

# DO loop

**DO** *index-variable=specification-1 <, ...specification-n> ;*  
*...more SAS statements...*

**END;**

- **index-variable**: nazwa zmiennej sterującej wykonaniem pętli.
- **specification**: wyrażenie definiujące liczbę iteracji pętli:  
*start <TO stop> <BY increment> <WHILE(expression) / UNTIL(expression)>*

# DO WHILE loop

**DO WHILE** (expression);  
...more SAS statements...

**END;**

- **(expression)**: wyrażenie logiczne umieszczone w nawiasach.
- Wyrażenie jest sprawdzane na początku pętli. Jeśli wyrażenie jest prawdziwe iteracja pętli się wykonuje. Jeśli wyrażenie jest fałszywe iteracji pętli się nie wykonuje, a wykonanie data stepu przechodzi do kolejnej instrukcji za pętlą. Pętla nie musi wykonać się ani razu.

# DO UNTIL

**DO UNTIL** (expression);  
...more SAS statements...

**END;**

- **(expression)**: wyrażenie logiczne umieszczone w nawiasach.
- Wyrażenie jest sprawdzane na końcu pętli. Jeśli wyrażenie jest fałszywe pętla wykona następną iterację. Jeśli wyrażenie jest prawdziwe pętla kończy działanie, a wykonanie data stepu przechodzi do kolejnej instrukcji za pętlą. Pętla musi wykonać się przynajmniej raz.



# Pętle – proste przykłady

- `do month='JAN','FEB','MAR';`
- `do count=2,3,5,7,11,13,17;`
- `do i=var1, var2, var3;`
- `do i=1 to 10;`
- `do i=1 to exit;`
- `do i=1 to x-5;`
- `do i=k+1 to n-1;`
- `do i=n to 1 by -1;`
- `do while(i<5);`
- `do until(j>=k);`

# Loops – złożone przykłady

- do i=1 to 10 while(x<y);
- do i=2 to 20 by 2 until((x/3)>y);
- do i=10 to 0 by -1 while(month='JAN');
- do i=.1 to .9 by .1, 1 to 10 by 1, 20 to 100 by 10;
- do i=1 to k-1, l+1 to n;
- do while(k<=**10** and l<=n);