# Cum utilizam -Functii Definite

## Initialization

```
ClearAll["Global`*"];
Off[General::spell, General::spell1]
```

Sa definim doua functii care aparent sunt identice:

```
gresit[x] := x + 5
adev[x_] := x + 5
```

sa le testam

```
gresit[x]
adev[x]
```

```
5 + x
```

```
5 + x
```

Totul pare OK- Care este problema?  Sa dam functiilor definite argu mente valorice:

```
gresit[2]
adev[2]
```

```
gresit[2]
```

```
7
```

Functia gresit nu stie ce sa faca cu argumentul 2 si Mathematica cand nu stie sa evalueze o expresie va returna forma sa originala.gresit[2]).  Sa incercam si cu argumente simbolice

```
gresit[y]
adev[y]
```

```
gresit[y]
```

```
5 + y
```

Putem extinde sintaxa pentru ca functia sa mearga numai pentru un tip particular de argument. Sa definim o functie care poate lucra numai cu argumente intregi.

```
adevinteger[x_Integer] := x + 5
```

```
adevinteger[1]
adevinteger[1.5]
```

```
6
```

```
adevinteger[1.5]
```

Devine clara posibilitatea definirii unor functii pentru anumite valori specifice ale argumentului lor:

```
si[x_] := Sin[x] / x
```

alegem

```
si[0.5]
```

```
0.958851
```

dar

```
si[0.0]
```

Power::infy : Infinite expression $\frac{1}{0.}$ encountered. ≫

∞::indet : Indeterminate expression 0. ComplexInfinity encountered. ≫

```
Indeterminate
```

Putem inlatura neajunsul

```
si[0.0] := 1.0
```

fixand punctul dorit fara a afecta celelalte cazuri

```
si[0.00]
si[0.5]
```

```
1.
```

```
0.958851
```

Sa aflam totul despre si

```
?? si
```

```
Global`si
```

```
si[0.] := 1.
```

$$si[x\_] := \frac{Sin[x]}{x}$$

sau pentru mai multe argumente,

$$myfunc2D[x\_, y\_] := x^2 y - 3 \frac{x^3}{y^2} + 8 y$$

Functia poate fi diferentiata

$$\partial_{x,y} myfunc2D[x, y]$$

$$2 x + \frac{18 x^2}{y^3}$$

Functia se poate dezvolta in serie Taylor in x=p

```
Series[myfunc2D[x, y], {x, p, 3}]
```

$$\left(-\frac{3\,p^3}{y^2}+8\,y+p^2\,y\right)+\left(-\frac{9\,p^2}{y^2}+2\,p\,y\right)(x-p)+\left(-\frac{9\,p}{y^2}+y\right)(x-p)^2-\frac{3\,(x-p)^3}{y^2}+O[x-p]^4$$

## Functii cu conditii

In *Mathematica* , uneori este necesara definirea unei functii care sa depinda de o conditie. Operatorul Conditie: *pattern /; test* poate fi utilizat in argumentul functiei pentru a deremina daca RHS trebuie evaluata.

```
myfunc3[x_ /; x > 0] := Sin[x]
myfunc3[x_ /; x ≤ 0] := Exp[3/10 x]
```

Sa dam valori diferite argumentului

```
myfunc3[3]
```

```
Sin[3]
```

```
myfunc3[-3]
```

$$\frac{1}{e^{9/10}}$$

Sa precizam ca functiile cu conditii nu pot fi integrate sau diferentiate

$$\left\{D[\text{myfunc3}[x], x], \int \text{myfunc2}[x]\,dx, \int_{-5}^{5} \text{myfunc2}[x]\,dx\right\}$$

$$\left\{\text{myfunc3}'[x], \int \text{myfunc2}[x]\,dx, \int_{-5}^{5} \text{myfunc2}[x]\,dx\right\}$$

dar, totusi integrarea numerica poate fi facuta

```
N[∫_{-5}^{5} myfunc3[x] ⅆx]
```

```
3.3059
```

## Functii pure

Functia ramane valabila in sistem atat timp cat nu este stearsa in mod explicit. De cele mai multe ori este mai bine ca functia sa fie definita o singura data si anume in momentul in care este neviue de ea.:

```
Sin[#1] + Cos[2 #1] &
```

```
Sin[#1] + Cos[2 #1] &
```

Aceasta este o functie pura , functie ce ia un singur argument, care este inserat pentru fie-care #. Functia pura poate fi recunoscuta prin prezenta lui & la sfarsit. Aceata poate fi apli-cata ca orice functie din Mathematica, argumentele sale fiind incluse intre paranteze drepte.

```
f[x_] := Sin[x] + Cos[2 x]
```

```
(Sin[#1] + Cos[2 #1] &)[3]
```

```
Cos[6] + Sin[3]
```

Spre exemplu: Utilizati #1, #2,.. pentru a defini o functie pura cu mai multe argumente:

```
(Sin[#1] + Cos[#2] &)[3, 5]
```

```
Cos[5] + Sin[3]
```

Functia pura Cos[#1] +Sin[#2] & este aplicata pe argumentele sale si abia apoi descarcata.

Daca se doreste un calcul cu argumente se utilizeaza aceleasi reguli de compunele obiectuala

```
(Sin[#1] + Cos[2 * #1 + #2] &)[3, 5]
```

```
Cos[11] + Sin[3]
```

```
f1[y_] := y^2 + 1
```

```
f1[2]
```

```
5
```

```
(f[#1] &)[2]
```

```
Cos[4] + Sin[2]
```

```
(f[#1] + f1[#2] &)[2, 3]
```

```
10 + Cos[4] + Sin[2]
```

```
fexp[x_] := Log[Cos[x]]
fpure = Log[Cos[#]] &
fexp[a]
fpure[a]
```

```
Log[Cos[#1]] &
```

```
Log[Cos[a]]
```

```
Log[Cos[a]]
```

sau pentru mai multe argumente,

```
fexp[x_, y_] := Cos[x] + Sin[y]
fpure = Cos[#1] + Sin[#2] &
fexp[a, b]
fpure[a, b]
```

```
Cos[#1] + Sin[#2] &
```

```
Cos[a] + Sin[b]
```

```
Cos[a] + Sin[b]
```

Sa precizam ca argumentele sunt culese in ordinea #1, #2, etc, iar & indica continutul inainte de a deveni functie.

```
#[[2]] & {{a, b}, {c, d}}
Map[#[[2]] &, {{a, b}, {c, d}}]
```

```
{{a (#1⟦2⟧ &), b (#1⟦2⟧ &)}, {c (#1⟦2⟧ &), d (#1⟦2⟧ &)}}
```

```
{b, d}
```

## Problem 1:

Define the following function in *Mathematica*: **f (x) =x²–Cos (x –x³)**. Then determine the following: `f (2)`, `f (t^2)`, `f (3.1)`. Before you begin this assignment Type in **`Remove[f]`** and then evaluate. This is to remove all previous definitions of functions called **`f`**.

### Problem 2:

Functions and variables that you type into an Input cell are stored with their definitions in the **Kernel**. You can see the definition by typing **?funcName**, where "**funcName**" is the name or symbol used to define your function or expression. The symbol **?** is the shorthand expression for the Information function. Use it to inspect the following variables: **f** and **f[x]**, **f[x_]**. What you should observe is that variable name is used for looking up the function, and not its functional dependence.

### Problem 3:

You can remove a definition from the kernel database (also know as the Global context; see the output from Problem 8.2) with the function **Remove[funcName]** . Use **Remove** to delete your definition of **f[x_]** and then use **?** to check whether it is still listed in the kernel database. Also try to evaluate **f[x]**.

### Problem 4:

Define a new function, say `g[x_]=Tan[x]` and evaluate the cell. Check the the definition using **?**. Evaluate `g[3.]`, `g[Sin[x]]`. Evaluate the latter at `x=π/2`. Next type in the following and evaluate: **?g**, `Clear[g]`. Then use **?** to check on the definition of **g** again. Note that the symbol **g** for your function is still listed in the kernel but the definition has been deleted. Now evaluate `Remove[g]` Use **?** to check on the definition. In summary, `Clear` deletes defintions from the kernel, while `Remove` deletes all refereneces to that variable or function.

## Problem 5:

Type in the following expression into an input cell: `f[x]:=Tan[x]`, and then evaluate the expressions `f[x]`, and `f[3]`.Compare the outcome of your result with Problem 8.4. Can you explain what is going on? (Caution: Be sure to remove all previous defintions of **f** before doing this problem, using the function `Remove`)

## Problem 6:

## Problem 7:

## Problem 8: