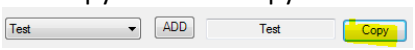
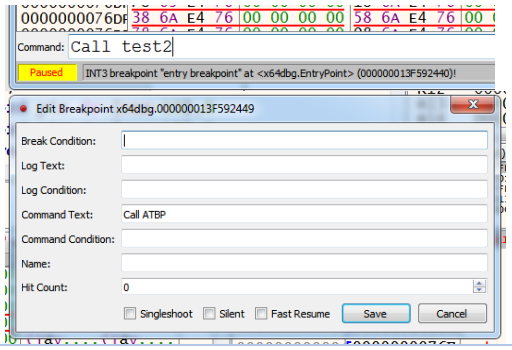
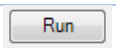
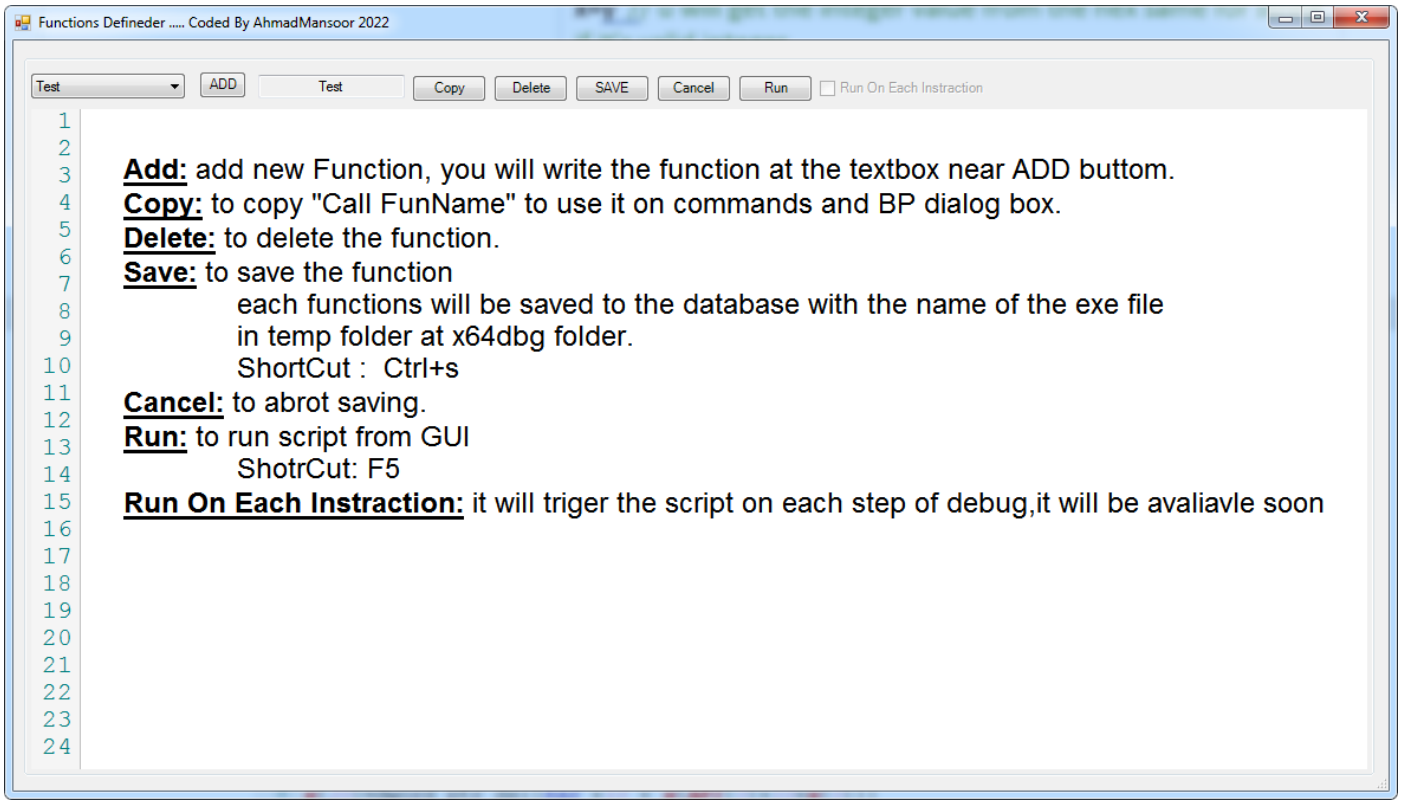
- **Summary:**
  - This is just a x64dbg script system support.
  - System export 2 functions: Call & Logs.
    Call FunctionsName:
    - Used to call the functions you already written in GUI.
    - Call can be standalone and run script from GUI or command line.

    logs input
    logs input,outpath
  - System use Fast Colored TextBox for GUI Syntax Highlighting and AutoComplete, Thanks for (Pavel Torgashov).
  - It supports most functions in x64dbg.
  - It's just to make your life easy that's all.

Notes:	
Main Function is Call	Call FunctionName Sample: Call Test Use Copy button to copy this line .  You can use this command from command line or on BP dialog box 
All number in the script window are in hex mode.	0x0000000000000030
‘ is the quoted symbol to use for quoted-string	Sample: ‘this is test’ use only ‘ **“ this will not replaced ‘ , so use ‘ for the strings** “ it will be like any other character
You can use + to joint strings to gather.	‘this value:’ + var + ‘same value as:’ + 10F U can add 2 string to gather Sample: Str x x=‘this is test’+’: its work’ **don’t left spaces between ‘ and +** logs x ret Result: this is test: its work But if u make spaces like this: x=‘this is test’ + ’: its work’ Result: this is test : its work
case sensitive	RAX is not same rax Please take care to write the words with case sensitive str not STR int not Int and so on
You can shift between types by assign variable to other	int x,10 hex y,25 x=y // u will get the integer value from the hex same for string if it’s valid integer.

	Logs x Result : x=37
F5 bottom	Used to run the script from the GUI same 
Ctrl + s	To save the script on GUI same 
ret	Each function should end with ret Script will stop at any line where ret exist ret can be at any line
Redefine Variable	U can redefine any variable in any step of the script, and u can change its type too.
Brackets	Any function begins at the beginning of the line not use bracket Other functions inside the command should use brackets Except python functions



- **Define variables:**

  This system supports this type of variables: int , intarr, str , strarr , hex , hexarr.

cmd	description	structure	sample	notes
int	normal integer number	int varname int varname,varvalue	int var int var,10F	In varvalue u can use functions or other variable value too like: int x,20 logs x hex y,x+10 logs y str z,y+x logs z ret Result: x=32 as integer 32 y=0x0000000000000030 as hex 0x0000000000000030 z=0x0000000000000050 as string value:0x0000000000000050
intarr	Array of normal integer number	intarr varname[size] intarr varname[size],first item value	intarr var[10] intarr var[10],10F	
str	string	str varname str varname,varvalue	str var str var,'this is sample'	
strarr	array of string	strarr varname[size] strarr varname[size], first item value	strarr var[10] strarr var[10],10F	
hex	integer number in hex mode	hex varname hex varname,varvalue	hex var hex var,10F	
hexarr	array of hex numbers	hexarr varname[size] hexarr varname[size], first item value	hexarr var[10] hexarr var[10],10F	

Sample:

```
int x,0000000077325701
str jj,'this is test'+$tid()+10
logs jj
jj=10+'Tet'+'heheh'+RAX
logs jj
```

- **Fast Access:**

System is support fast access to registers and memory direct commands

- Register:

Assign value to register: REG=any hex value

```
hex x,25f
hexarr y[10],30a
RAX=RAX+1  // change RAX value by add 1
EBX=y[0]+x-AX // assign EBX value usin x varible and y array and AX register
ret
```

Read register value: variable=REG

```
hex x,25f
hexarr y[10],30a
y[20]=RAX+1  // assign record 20 of Array y
x=y[0]+x-AX // assign x value usin y array and AX register
ret
```

```
hexarr x[25],10
strarr API[3],1
x[10]=$ArrayLen(x) + $ArrayLen(x) + 20
logs x[10]
API[2]=$ReadStr($mod.base(RIP)+3000+398)+$ReadStr($mod.base(RIP) +3000+ 398)
logs API[2]
RAX=qword ptr ds:[RAX+word ptr ds:[AX+10]]
x[20]=dword ptr ds:[RAX +10 + x[API[0]+10+x[0]]]
logs 'x[20]=' + x[20]
API[2]=10 + x[API[0]+10+x[0]]
logs API[2]
x[2]=x[0]+25
logs x[2]
ret
```

- Memory:

Assign value to memory directly or read memory to variable:

memory = hex value

var=memory

```
hex x
strarr y[10],30
y[2]='C868E4760000000002869E47600000000'
// read memory to variable
x=byte ptr ds:[RAX+1]
x=word ptr ds:[RAX - RBX]-byte ptr ds:[RDI]
x=dword ptr ds:[R8]
x=qword ptr ds:[RAX + byte ptr ds:[RAX]]
// write to memory
byte ptr ds:[RAX]=25
word ptr ds:[RAX]=RAX-1
dword ptr ds:[RBX]=y[0]+byte ptr ds:[RSI]
qword ptr ds:[R8+x]=byte ptr ds:[RAX]+9A10
// write n hex value to memory
[RAX]='C868E4760000000002869E47600000000'
//or
[RAX]=y[2]
ret
```

- **Condition command:**

If command, syntax

If.cmdNumber condition ..... End.cmdNumber	If.cmdNumber condition ..... Else.cmdNumber ..... End.cmdNumber	If.cmdNumber condition ..... repeat.cmdNumber End.cmdNumber

Sample:

```
str x,'This is Test'
hex address,0000000076DF1000
int i,0  // counter
strarr y[10]
// if end
if.1 RAX >=0
logs RAX
End.1

// if else end
if.2 x==$ReadStr(address)
logs 'This is Test'
Else.2
logs  'Not equal'
End.2

if.3 CIP!=RAX
logs 'CIP=' + CIP
Else.3
logs 'RAX=' + RAX
End.3

// if repeat end,can be used as loop like to fill array items
if.4 i< $ArrayLen(y)
y[i]='RAX='+RAX
i=i+1
repeat.4
End.4
ret
```

- **Commands Supports:**

```
static array <String^>^ x64dbgFunList = gcnew array <String^> { "run ", "erun ", "serun ", "pause ", "DebugContinue ",
    "StepInto ", "sti ", "eStepInto ", "esti ", "seStepInto ", "sesti ", "StepOver ", "eStepOver "
    , "seStepOver ", "StepOut ", "eStepOut ", "skip ", "InstrUndo ",
    "bp ", "bph ", "bpm ", "bpdll ", "bpdll ", "bpdll ", "SetExceptionBPX ", "bpgoto ",
    "bpgoto ", "bpc ", "bpe ", "bpd ", "bpc ", "bpe ", "bpd ", "bphc ", "bphe ", "bphd ",
    "bpmc ", "bpme ", "bpmd ", "bcdll ", "bpedll ",
    "bpddll ", "DeleteExceptionBPX ", "EnableExceptionBPX ", "DisableExceptionBPX ", "SetBPXOptions ",
    "SetBreakpointCondition ","SetBreakpointLog ", "SetBreakpointLogCondition ", "SetBreakpointCommand ",
    "SetBreakpointCommandCondition ", "SetBreakpointFastResume ","SetBreakpointSingleshoot ", "SetBreakpointSilent ",
    "ResetBreakpointHitCount ", "SetHardwareBreakpointCondition ", "SetHardwareBreakpointLog ",
    "SetHardwareBreakpointLogCondition ", "SetHardwareBreakpointCommand ", "SetHardwareBreakpointCommandCondition ",
    "SetHardwareBreakpointSilent ", "SetHardwareBreakpointSingleshoot ", "SetHardwareBreakpointFastResume ",
    "SetMemoryBreakpointLog ", "SetMemoryBreakpointCondition ","ResetHardwareBreakpointHitCount ",
    "SetMemoryBreakpointCommandCondition ", "SetMemoryBreakpointCommand ", "SetMemoryBreakpointLogCondition ",
    "SetMemoryBreakpointSilent ", "SetMemoryBreakpointSingleshoot ", "SetMemoryBreakpointFastResume ",
    "SetLibrarianBreakpointLog ", "SetLibrarianBreakpointCondition ",
    "ResetMemoryBreakpointHitCount ", "SetLibrarianBreakpointCommandCondition ","SetLibrarianBreakpointCommand ",
    "SetLibrarianBreakpointLogCondition ","SetLibrarianBreakpointSilent ", "SetLibrarianBreakpointSingleshoot ",
    "SetLibrarianBreakpointFastResume ", "SetExceptionBreakpointLog ","SetExceptionBreakpointCondition ",
    "ResetLibrarianBreakpointHitCount ", "SetExceptionBreakpointCommandCondition ", "SetExceptionBreakpointCommand ",
    "SetExceptionBreakpointLogCondition ", "SetExceptionBreakpointSilent ", "SetExceptionBreakpointSingleshoot ",
    "SetExceptionBreakpointFastResume ","ResetExceptionBreakpointHitCount ", "createthread ", "suspendthread ",
    "switchthread ", "resumethread ", "suspendallthreads ", "killthread ","setthreadname ", "setthreadpriority ",
    "resumeallthreads ", "alloc", "memset ", "memcpy ", "free ", "setpagerights ", "savedata ", "DisablePrivilege ",
    "EnablePrivilege ", "handleclose ", "dbsave ", "dbload ", "dbclear ", "commentset ", "commentdel ", "commentclear ",
    "labelset ","labeldel ", "labelclear ", "bookmarkset ", "bookmarkdel ", "bookmarkclear ", "functionadd ",
    "functiondel ", "functionclear ", "argumentadd ","argumentdel ", "argumentclear ", "findall ",
    "findallmem ", "findasm ", "findref ", "strref ", "modcallfind " , "setmaxfindresult ","doSleep ",
    "HideDebugger", "loadlib ", "asm ","setcmdline ","Fill ","logs ", "pythonBase " , "py_define ","$ResizeArray "
};
```

You can find reference for all Commands at x64dbg help site:

https://help.x64dbg.com/en/latest/commands/index.html

```
asm CIP,'mov rcx, qword ptr gs:[0x0000000000000030]',0
ret
```

```
strarr x[10],'byte ptr ds:['+RAX+']'
hexarr y[3]
str fl
intarr v[16],25
y[0]=RAX + AX+EBX - byte ptr ds:[RAX]+25
bp RAX,'"Test"',ss
bph $mod.base(CIP)+390E
fl=$dis.len(RIP)
logs 'First excption' + fl
ret
```

```
hexarr IATTable[1],RAX
IATTable[0]=IATTable[0]+10
logs IATTable
//RAX=IATTable[0]+10
logs IATTable[0]
Fill IATTable[0],25
dword ptr ds:[IATTable[0]]= 105
ret
```

- **Extra Commands:**

1- Logs: log anything u want like variables or anything

Syntax :

logs anything

or

logs anything, path(to save to file)

```
logs RAX,'C:\test.txt'
```

Sample:

```
strarr x[10],'byte ptr ds:['+RAX+']'
hexarr y[3]
str fl
intarr v[16],25
y[0]=RAX + AX+EBX - byte ptr ds:[RAX]+25
bp RAX,'"Test"',ss
//bpc '"Test"'
fl=$dis.len(RIP)
logs 'First excption' + fl
ret
```

2- $ commands: it's two parts x64dbg part and script part

- x64dbg parts:

```
static array <String^>^ scListConst = { "$disasm.sel()" , "$dump.sel()","$stack.sel()","$peb()","$teb()","$tid()","$mod.main()",
    "$GetTickCount()" ,"$func.start()" , "$func.end()" , "$ref.count()" ,"$ex.firstchance()" , "$ex.addr()" ,
    "$ex.code()" , "$ex.flags()","$ex.infocount()","$result"};

static array <String^>^ scListpara = { "$src.disp()" , "$src.line()","$mod.party()","$mod.base()","$mod.size()",
    "$mod.hash()","$mod.entry()","$mod.system()" ,"$mod.user()" , "$mod.rva()" , "$mod.offset()" ,"$mod.isexport()" ,
    "$bswap()" , "$utf8()" , "$utf16()","$mem.valid()","$mem.base()","$mem.size()" ,"$mem.iscode()" ,"$dis.len()" ,
    "$dis.iscond()","$dis.isbranch()","$dis.isret()","$dis.iscall()","$dis.ismem()","$dis.isnop()","$dis.isunusual()" ,
    "$dis.branchdest()" ,"$dis.branchexec()" ,"$dis.imm()","$dis.brtrue()","$dis.brfalse()","$dis.next()","$dis.prev()",
    "$dis.iscallsystem()" };
```

- `scListConst : commands not take argument's.`
- `scListpara : commands take one argument.`

We got them by typing $ and the Auto List will show them.

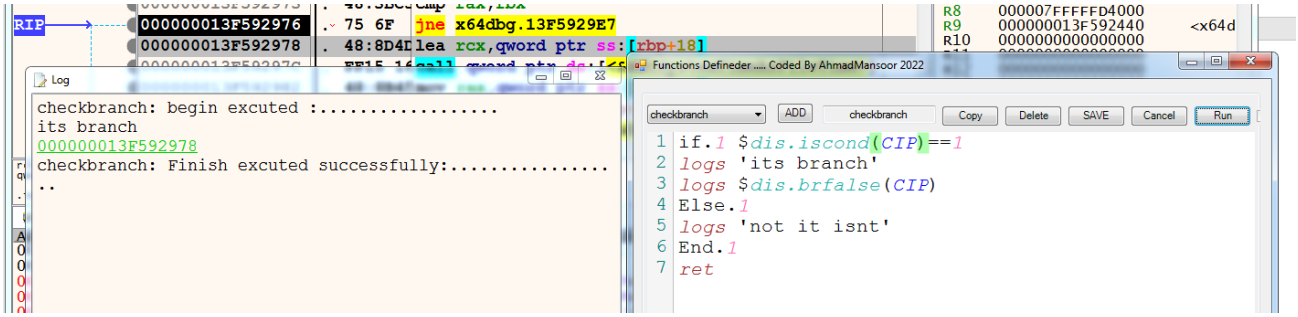
reference:

https://help.x64dbg.com/en/latest/introduction/Expression-functions.html?highlight=mod.entry#modules

sample:

```
hex ss,15
str Val,10
strarr gg[2],15
str Val,30
if.1 gg[0]==15
ss= $peb()
ss = $mod.base($mod.entry(CIP))
logs ss
logs 'before else'
Else.1
ss=$dump.sel()
logs ss
ss=$peb()
logs ss
logs 'after else'
End.1
ss=$teb() + $dump.sel()
//ss=qword ptr ds:[RAX]
logs ss
ret
```

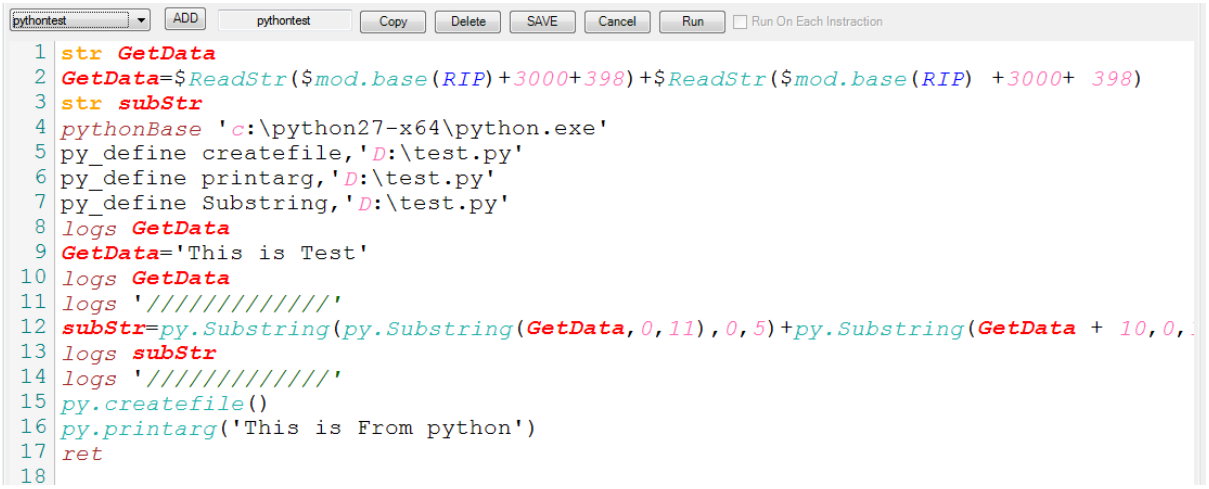
We can use the condition commands too like check if branch or not.



- script part: this extra function it could modify later according to the needed of the users:

```
static array <String^>^ scListextra = { "$ReadStr()", "$ArrayLen()" };
// ReadStr(Address)      ResizeArray(VarArray)
```

$ReadStr: will enable u to read string at address if valid.



$ArrayLen: function to resize the array size u can increase or decrease.
Sample:

ArrayLen(arrayVar,1) //increase arrayVar by 1
ArrayLen(arrayVar, FFFFFFFFFFFFFFFF) // decrease arrayVar by 1

FFFFFFFFFFFFFFFF=-1 in x64 system
FFFFFFFF=-1 in x32 system so take care when u write sign values.

```
24 if.4 i< $ArrayLen(y)
25 y[i]='RAX='+RAX
26 i=i+1
27 repeat.4
28 End.4
29 ret
```

- **Python:**

Now you can define python commands and use them in the script direct, you can call function with many arguments.
Limitation: you can get one value as return value.
It will support array return later.
How it works:

- pythonBase : it used to define the path of the python u used , it should defined at the top before any call.
  pythonBase pythonPath
  ```
  pythonBase 'c:\python27-x64\python.exe'
  ```
- py_define: it will define the function u want to call and script path.
  Syntax : py_define FunctionName,ScriptPath
  ```
  py_define createfile,'D:\test.py'
  ```
- py : it used to call the function that all ready define it in py_define.
  U can call functions anywhere in the command or as standalone function.
  Sample:

```
str GetData
GetData=$ReadStr($mod.base(RIP)+3000+398)+$ReadStr($mod.base(RIP) +3000+ 398)
str subStr
pythonBase 'c:\python27-x64\python.exe'
py_define createfile,'D:\test.py'
py_define printarg,'D:\test.py'
py_define Substring,'D:\test.py'
logs GetData
GetData='This is Test'
logs GetData
logs '/////////////'
subStr=py.Substring(py.Substring(GetData,0,11),0,5)+py.Substring(GetData + 10,0,18)
logs subStr
logs '/////////////'
py.createfile()
py.printarg('This is From python')
ret
```

Python script structure should be similar like this:

```
import sys
def pythonMessage(name):
    print ("printarg: " +  name )
def createfile():
    f  = open("D:\\test.txt", "w+")
    f.write("test")
    f.close()
def printme():
    pythonMessage(sys.argv[1])
def printarg(arg):
    print ("printarg: " +  arg )
    f  = open("D:\\test1.txt", "w+")
    f.write(arg)
    f.close()
def Substring():
    start=int(sys.argv[3])
    end=int(sys.argv[4])
    print (sys.argv[2][start:end])
if sys.argv[1] == "printme" :
    printme()
if sys.argv[1] == "printarg" :
    printarg(sys.argv[2])
if sys.argv[1] == "createfile" :
    createfile()
if sys.argv[1] == "Substring" :
    Substring()
```

argv[0] : script path
argv[1] : function name
argv[.....] : arguments of the functions

Hotkeys

The control supports following hotkeys:

- Left, Right, Up, Down, Home, End, PageUp, PageDown - moves caret
- Shift+(Left, Right, Up, Down, Home, End, PageUp, PageDown) - moves caret with selection
- Ctrl+F, Ctrl+H - shows Find and Replace dialogs
- F3 - find next
- Ctrl+G - shows GoTo dialog
- Ctrl+(C, V, X) - standard clipboard operations
- Ctrl+A - selects all text
- Ctrl+Z, Alt+Backspace, Ctrl+R - Undo/Redo opertions
- Tab, Shift+Tab - increase/decrease left indent of selected range
- Ctrl+Home, Ctrl+End - go to first/last char of the text
- Shift+Ctrl+Home, Shift+Ctrl+End - go to first/last char of the text with selection
- Ctrl+Left, Ctrl+Right - go word left/right
- Shift+Ctrl+Left, Shift+Ctrl+Right - go word left/right with selection
- Ctrl+-, Shift+Ctrl+- - backward/forward navigation
- Ctrl+U, Shift+Ctrl+U - converts selected text to upper/lower case
- Ctrl+Shift+C - inserts/removes comment prefix in selected lines
- Ins - switches between Insert Mode and Overwrite Mode
- Ctrl+Backspace, Ctrl+Del - remove word left/right
- Alt+Mouse, Alt+Shift+(Up, Down, Right, Left) - enables column selection mode
- Alt+Up, Alt+Down - moves selected lines up/down
- Shift+Del - removes current line
- Ctrl+B, Ctrl+Shift-B, Ctrl+N, Ctrl+Shift+N - add, removes and navigates to bookmark
- Esc - closes all opened tooltips, menus and hints
- Ctrl+Wheel - zooming
- Ctrl+M, Ctrl+E - start/stop macro recording, executing of macro
- Alt+F [char] - finds nearest [char]
- Ctrl+(Up, Down) - scrolls Up/Down
- Ctrl+(NumpadPlus, NumpadMinus, 0) - zoom in, zoom out, no zoom
- Ctrl+I - forced AutoIndentChars of current line