# IDAPython脚本化软件逆向分析

## 课前准备

1. 学员基础：
   1. 熟悉C/C++
   2. 熟悉x86汇编
   3. 熟悉Python
   4. 具备《**SQLite数据库逆向分析**》
   5. 具备《x86/x64软件逆向分析》更佳
   6. 具备《X86软件逆向分析实战》更佳
2. 演示环境：
   1. Windows 10
   2. Visual Studio Community 2022 /C++桌面开发

<https://visualstudio.microsoft.com/zh-hans/>

* 1. IDA Pro 7.7

<https://hex-rays.com/>

<https://hex-rays.com/ida-pro/>

* 1. WeChat SQLite

<https://pc.weixin.qq.com/>

* 1. PyCharm Community 2022.1

<https://www.jetbrains.com.cn/en-us/pycharm/download/#section=windows>

<https://download.jetbrains.com.cn/python/pycharm-community-2022.1.exe>

* 1. Wing Pro 8

<http://wingware.com/downloads/wingide>

<https://wingware.com/pub/wingpro/8.3.0.1/wingpro-8.3.0.1.exe>

1. 课程源码：[https://github.com/**zmrbak**/IDAPython](https://github.com/zmrbak/IDAPython)
2. 课程类型：演示、实战

## 源码编译常见问题及解决

**升级项目**

右击项目，属性，

配置（所有配置），平台（所有平台），

配置属性，常规，

选择Windows SDK 版本，选择平台工具集，

确定。

**编译**

配置：debug/release

平台: x86

右击项目，生成/重新生成

## MSVC编译控制参数(1)

<https://docs.microsoft.com/zh-cn/cpp/build/projects-and-build-systems-cpp?view=msvc-170>

|  |  |
| --- | --- |
| **Debug** | **Release** |
| **/JMC**  启用本机“仅我的代码”  **/permissive-**  使某些非符合代码可编译(功能集可更改)(默认开启)  **/ifcOutput "Debug\"**  建议使用 /ifcOutput <directory> 为每个编译创建一个单独 .ifc 的文件。  **/GS**  启用安全检查  **/analyze-**  启用本机分析  **/W3**  设置警告等级(默认 n=1)    **/Zc:wchar\_t**  C++ 语言合规性，这里的参数可以是: wchar\_t 是本机类型，不是 typedef  **/ZI**  启用“编辑并继续”调试信息  **/Gm-**  启用最小重新生成  **/Od**  禁用优化(默认)  **/sdl**  支持其他安全功能和警告  **/Fd"Debug\vc143.pdb"**  命名 .PDB 文件  **/Zc:inline**  C++ 语言合规性，这里的参数可以是  如果是 COMDAT，则删除未引用的函数或数据, 或仅使用内部链接(默认关闭)  **/fp:precise**  选择浮点模型, "precise" 浮点模型；结果可预测  **/D "WIN32"**  定义宏  **/D "\_DEBUG"**  **/D "\_CONSOLE"**  **/D "\_UNICODE"**  **/D "UNICODE"**  **/errorReport:prompt**  已弃用。请将内部编译器错误报告给 Microsoft  prompt - 提示立即发送报告  **/WX-**  将警告视为错误  **/Zc:forScope**  C++ 语言合规性，这里的参数可以是:  对范围规则强制使用标准 C++  **/RTC1**  启用快速检查(/RTCsu)  **/Gd**  \_\_cdecl 调用约定  **/Oy-**  启用帧指针省略  **/MDd**  与 MSVCRTD.LIB 调试库链接  **/FC**  诊断中使用完整路径名  **/Fa"Debug\"**  命名程序集列表文件  **/EHsc**  GX[-] 启用 C++ EH (与 /EHsc 相同)  EHs 启用 C++ EH (没有 SEH 异常)  EHc 外部 "C" 默认为 nothrow  **/nologo**  取消显示版权信息  **/Fo"Debug\"**  命名对象文件  **/Fp"Debug\ida03.pch"**  命名预编译头文件  **/diagnostics:column**  控制诊断消息的格式：打印列信息 | **/permissive-**  使某些非符合代码可编译(功能集可更改)(默认开启)  **/ifcOutput "Release\"**  建议使用 /ifcOutput <directory> 为每个编译创建一个单独 .ifc 的文件。  **/GS**  启用安全检查  **/GL**  启用链接时代码生成  **/analyze-**  启用本机分析  **/W3**  设置警告等级(默认 n=1)  **/Gy**  分隔链接器函数  **/Zc:wchar\_t**  C++ 语言合规性，这里的参数可以是: wchar\_t 是本机类型，不是 typedef  **/Zi**  启用调试信息  **/Gm-**  启用最小重新生成  **/O2**  最大优化(优选速度)  **/sdl**  支持其他安全功能和警告  **/Fd"Release\vc143.pdb"**  命名 .PDB 文件  **/Zc:inline**  C++ 语言合规性，这里的参数可以是  如果是 COMDAT，则删除未引用的函数或数据, 或仅使用内部链接(默认关闭)  **/fp:precise**  选择浮点模型, "precise" 浮点模型；结果可预测  **/D "WIN32"**  定义宏  **/D "NDEBUG"**  **/D "\_CONSOLE"**  **/D "\_UNICODE"**  **/D "UNICODE"**  **/errorReport:prompt**  已弃用。请将内部编译器错误报告给 Microsoft  prompt - 提示立即发送报告  **/WX-**  将警告视为错误  **/Zc:forScope**  C++ 语言合规性，这里的参数可以是:  对范围规则强制使用标准 C++  **/Gd**  \_\_cdecl 调用约定  **/Oy-**  启用帧指针省略  **/Oi**  启用内部函数  **/MD**  与 MSVCRT.LIB 链接  **/FC**  诊断中使用完整路径名  **/Fa"Release\"**  命名程序集列表文件  **/EHsc**  GX[-] 启用 C++ EH (与 /EHsc 相同)  EHs 启用 C++ EH (没有 SEH 异常)  EHc 外部 "C" 默认为 nothrow  **/nologo**  取消显示版权信息  **/Fo"Release\"**  命名对象文件  **/Fp"Release\ida03.pch"**  命名预编译头文件  **/diagnostics:column**  控制诊断消息的格式：打印列信息 |

%comspec% /k "C:\Program Files\Microsoft Visual Studio\2022\Preview\Common7\Tools\VsDevCmd.bat"

## MSVC编译控制参数(2)

## IDA Pro中的脚本插件（1）

目录结构

cfg 配置文件

dbgsrv 动态调式所需的server文件

**idc** 内置脚本语言IDC所需的核心文件

ids 描述加载的二进制文件共享库内容

loaders 用于加载识别解析文件格式的扩展文件

platforms 平台相关的窗口库

plugins 扩展插件的文件夹

procs 处理器支持模块，机器码转汇编

python IDAPython库

https://github.com/idapython/src

python38 IDA内置的Python程序

sig 模式匹配的签名

themes IDA外观样式

til 类型库信息

github树状插件

<https://www.octotree.io/>

IDAPython API随机文档

IDA\_Pro\_7.7/python/examples/index.html

IDA\_Pro\_7.7/idahelp.chm

启动IDA时Python插件出错

为PATH添加Python路径 IDA\_Pro\_7.7\python38

## IDA Pro中的脚本插件（2）

## 程序的Debug/Release以及符号文件对IDA的影响（1）

编译器会生成与应用程序同名的PDB文件，作为符号文件，有助于IDA对程序中的函数名进行识别。

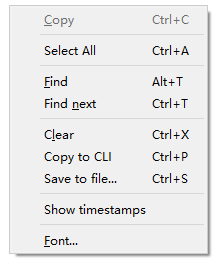
Debug模式中，编译器为程序添加了一些额外的调试代码，有助于对程序的调试。

Release模式中，编译器对程序做了一些优化，一些函数被内联到其它函数之中，未被引用到的函数会在链接阶段被移除。

## 程序的Debug/Release以及符号文件对IDA的影响（2）

## 使用Wing Pro调试IDAPython脚本

Output窗口快捷键



Wing Pro 调试设置

import wingdbstub

import idc

import idautils

wingdbstub.Ensure()

print('Test From Wing Pro 8')

for seg in idautils.Segments():

name = idc.get\_segm\_name(seg)

start = idc.get\_segm\_start(seg)

end = idc.get\_segm\_end(seg)

print(name,start,end)

接受调试连接

开始调试

IDA中，运行脚本

wingdbstub.py

C:\Program Files (x86)\Wing Pro 8

## 使用PyCharm开发IDAPython脚本

运行脚本的方式

Shift+F2 打开脚本编辑窗口，粘贴代码，点击运行。

将代码粘贴到python cli，按两次回车

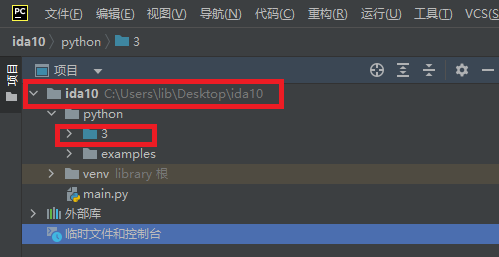
Alt+F7，选择脚本文件，执行脚本文件

PyCharm配置

创建项目

将IDA\_Pro\_7.7\python包复制到项目中

将IDA\_Pro\_7.7\python包中的3设置为“源 根”



## 了解IDAPython中的内置函数

idc.py 使用Python封装了的IDC模块

idautils.py IDA中高级实用得到功能模块

idaapi.py IDA中允许访问更低层数据的模块

列出python文件中的函数

E:\IDA\_Pro\_7.7\python\3>type idaapi.py | find "def "

def \_\_init\_\_(self):

def \_get\_module\_cvar(self, modname):

def \_\_getattr\_\_(self, attr):

def \_\_setattr\_\_(self, attr, value):

E:\IDA\_Pro\_7.7\python\3>type idc.py | find "def " >idc.txt

def \_\_warn\_once\_deprecated\_proto\_confusion(what, alternative):

def \_IDC\_GetAttr(obj, attrmap, attroffs):

def \_IDC\_SetAttr(obj, attrmap, attroffs, value):

def has\_value(F): return ((F & FF\_IVL) != 0) # any defined value?

def byte\_value(F):

def is\_loaded(ea):

def is\_code(F): return ((F & MS\_CLS) == FF\_CODE) # is code byte?

def is\_data(F): return ((F & MS\_CLS) == FF\_DATA) # is data byte?

def is\_tail(F): return ((F & MS\_CLS) == FF\_TAIL) # is tail byte?

def is\_unknown(F): return ((F & MS\_CLS) == FF\_UNK) # is unexplored byte?

def is\_head(F): return ((F & FF\_DATA) != 0) # is start of code/data?

def is\_flow(F): return ((F & FF\_FLOW) != 0)

def isExtra(F): return ((F & FF\_LINE) != 0)

def isRef(F): return ((F & FF\_REF) != 0)

def hasName(F): return ((F & FF\_NAME) != 0)

def hasUserName(F): return ((F & FF\_ANYNAME) == FF\_NAME)

def is\_defarg0(F): return ((F & MS\_0TYPE) != FF\_0VOID)

def is\_defarg1(F): return ((F & MS\_1TYPE) != FF\_1VOID)

def isDec0(F): return ((F & MS\_0TYPE) == FF\_0NUMD)

def isDec1(F): return ((F & MS\_1TYPE) == FF\_1NUMD)

def isHex0(F): return ((F & MS\_0TYPE) == FF\_0NUMH)

def isHex1(F): return ((F & MS\_1TYPE) == FF\_1NUMH)

def isOct0(F): return ((F & MS\_0TYPE) == FF\_0NUMO)

def isOct1(F): return ((F & MS\_1TYPE) == FF\_1NUMO)

def isBin0(F): return ((F & MS\_0TYPE) == FF\_0NUMB)

def isBin1(F): return ((F & MS\_1TYPE) == FF\_1NUMB)

def is\_off0(F): return ((F & MS\_0TYPE) == FF\_0OFF)

def is\_off1(F): return ((F & MS\_1TYPE) == FF\_1OFF)

def is\_char0(F): return ((F & MS\_0TYPE) == FF\_0CHAR)

def is\_char1(F): return ((F & MS\_1TYPE) == FF\_1CHAR)

def is\_seg0(F): return ((F & MS\_0TYPE) == FF\_0SEG)

def is\_seg1(F): return ((F & MS\_1TYPE) == FF\_1SEG)

def is\_enum0(F): return ((F & MS\_0TYPE) == FF\_0ENUM)

def is\_enum1(F): return ((F & MS\_1TYPE) == FF\_1ENUM)

def is\_manual0(F): return ((F & MS\_0TYPE) == FF\_0FOP)

def is\_manual1(F): return ((F & MS\_1TYPE) == FF\_1FOP)

def is\_stroff0(F): return ((F & MS\_0TYPE) == FF\_0STRO)

def is\_stroff1(F): return ((F & MS\_1TYPE) == FF\_1STRO)

def is\_stkvar0(F): return ((F & MS\_0TYPE) == FF\_0STK)

def is\_stkvar1(F): return ((F & MS\_1TYPE) == FF\_1STK)

def is\_byte(F): return (is\_data(F) and (F & DT\_TYPE) == FF\_BYTE)

def is\_word(F): return (is\_data(F) and (F & DT\_TYPE) == FF\_WORD)

def is\_dword(F): return (is\_data(F) and (F & DT\_TYPE) == FF\_DWORD)

def is\_qword(F): return (is\_data(F) and (F & DT\_TYPE) == FF\_QWORD)

def is\_oword(F): return (is\_data(F) and (F & DT\_TYPE) == FF\_OWORD)

def is\_tbyte(F): return (is\_data(F) and (F & DT\_TYPE) == FF\_TBYTE)

def is\_float(F): return (is\_data(F) and (F & DT\_TYPE) == FF\_FLOAT)

def is\_double(F): return (is\_data(F) and (F & DT\_TYPE) == FF\_DOUBLE)

def is\_pack\_real(F): return (is\_data(F) and (F & DT\_TYPE) == FF\_PACKREAL)

def is\_strlit(F): return (is\_data(F) and (F & DT\_TYPE) == FF\_STRLIT)

def is\_struct(F): return (is\_data(F) and (F & DT\_TYPE) == FF\_STRUCT)

def is\_align(F): return (is\_data(F) and (F & DT\_TYPE) == FF\_ALIGN)

def value\_is\_string(var): raise NotImplementedError("this function is not needed in Python")

def value\_is\_long(var): raise NotImplementedError("this function is not needed in Python")

def value\_is\_float(var): raise NotImplementedError("this function is not needed in Python")

def value\_is\_func(var): raise NotImplementedError("this function is not needed in Python")

def value\_is\_pvoid(var): raise NotImplementedError("this function is not needed in Python")

def value\_is\_int64(var): raise NotImplementedError("this function is not needed in Python")

def to\_ea(seg, off):

def form(format, \*args):

def substr(s, x1, x2):

def strstr(s1, s2):

def strlen(s):

def xtol(s):

def atoa(ea):

def ltoa(n, radix):

def atol(s):

def rotate\_left(value, count, nbits, offset):

def rotate\_dword(x, count): return rotate\_left(x, count, 32, 0)

def rotate\_word(x, count): return rotate\_left(x, count, 16, 0)

def rotate\_byte(x, count): return rotate\_left(x, count, 8, 0)

def eval\_idc(expr):

def EVAL\_FAILURE(code):

def save\_database(idbname, flags=0):

def validate\_idb\_names(do\_repair = 0):

def call\_system(command):

def qsleep(milliseconds):

def delete\_all\_segments():

def plan\_and\_wait(sEA, eEA, final\_pass=True):

def set\_name(ea, name, flags=ida\_name.SN\_CHECK):

def make\_array(ea, nitems):

def create\_strlit(ea, endea):

def create\_byte(ea):

def create\_word(ea):

def create\_dword(ea):

def create\_qword(ea):

def create\_oword(ea):

def create\_yword(ea):

def create\_float(ea):

def create\_double(ea):

def create\_pack\_real(ea):

def create\_tbyte(ea):

def create\_struct(ea, size, strname):

def define\_local\_var(start, end, location, name):

def set\_array\_params(ea, flags, litems, align):

def op\_plain\_offset(ea, n, base):

def toggle\_bnot(ea, n):

def op\_stroff(ea, n, strid, delta):

def op\_offset\_high16(ea, n, target):

def MakeVar(ea):

def split\_sreg\_range(ea, reg, value, tag=SR\_user):

def AutoMark(ea,qtype):

def gen\_file(filetype, path, ea1, ea2, flags):

def gen\_flow\_graph(outfile, title, ea1, ea2, flags):

def gen\_simple\_call\_chart(outfile, title, flags):

def idadir():

def get\_idb\_path():

def get\_bytes(ea, size, use\_dbg = False):

def \_\_DbgValue(ea, len):

def read\_dbg\_byte(ea):

def read\_dbg\_word(ea):

def read\_dbg\_dword(ea):

def read\_dbg\_qword(ea):

def write\_dbg\_memory(ea, data):

def GetFloat(ea):

def GetDouble(ea):

def get\_name\_ea\_simple(name):

def get\_segm\_by\_sel(base):

def get\_curline():

def read\_selection\_start():

def read\_selection\_end():

def get\_sreg(ea, reg):

def next\_head(ea, maxea=BADADDR):

def prev\_head(ea, minea=0):

def get\_item\_size(ea):

def func\_contains(func\_ea, ea):

def get\_name(ea, gtn\_flags=0):

def demangle\_name(name, disable\_mask):

def generate\_disasm\_line(ea, flags):

def GetDisasm(ea):

def print\_insn\_mnem(ea):

def print\_operand(ea, n):

def get\_operand\_type(ea, n):

def get\_operand\_value(ea, n):

def get\_strlit\_contents(ea, length = -1, strtype = STRTYPE\_C):

def get\_str\_type(ea):

def find\_text(ea, flag, y, x, searchstr):

def find\_binary(ea, flag, searchstr, radix=16):

def process\_config\_line(directive):

def \_import\_module\_flag\_sets(module, prefixes):

def get\_inf\_attr(attr):

def set\_inf\_attr(attr, value):

def SetPrcsr(processor): return set\_processor\_type(processor, SETPROC\_USER)

def batch(batch):

def process\_ui\_action(name, flags=0):

def sel2para(sel):

def find\_selector(val):

def get\_first\_seg():

def get\_next\_seg(ea):

def get\_segm\_start(ea):

def get\_segm\_end(ea):

def get\_segm\_name(ea):

def add\_segm\_ex(startea, endea, base, use32, align, comb, flags):

def AddSeg(startea, endea, base, use32, align, comb):

def set\_segment\_bounds(ea, startea, endea, flags):

def set\_segm\_name(ea, name):

def set\_segm\_class(ea, segclass):

def set\_segm\_alignment(ea, alignment):

def set\_segm\_combination(segea, comb):

def set\_segm\_addressing(ea, bitness):

def selector\_by\_name(segname):

def set\_default\_sreg\_value(ea, reg, value):

def set\_segm\_type(segea, segtype):

def get\_segm\_attr(segea, attr):

def set\_segm\_attr(segea, attr, value):

def move\_segm(ea, to, flags):

def get\_xref\_type():

def fopen(f, mode):

def fclose(handle):

def filelength(handle):

def fseek(handle, offset, origin):

def ftell(handle):

def LoadFile(filepath, pos, ea, size):

def loadfile(filepath, pos, ea, size): return LoadFile(filepath, pos, ea, size)

def SaveFile(filepath, pos, ea, size):

def savefile(filepath, pos, ea, size): return SaveFile(filepath, pos, ea, size)

def fgetc(handle):

def fputc(byte, handle):

def fprintf(handle, format, \*args):

def readshort(handle, mostfirst):

def readlong(handle, mostfirst):

def writeshort(handle, word, mostfirst):

def writelong(handle, dword, mostfirst):

def readstr(handle):

def writestr(handle, s):

def get\_next\_func(ea):

def get\_prev\_func(ea):

def get\_func\_attr(ea, attr):

def set\_func\_attr(ea, attr, value):

def get\_func\_flags(ea):

def set\_func\_flags(ea, flags):

def get\_func\_name(ea):

def get\_func\_cmt(ea, repeatable):

def set\_func\_cmt(ea, cmt, repeatable):

def choose\_func(title):

def get\_func\_off\_str(ea):

def find\_func\_end(ea):

def get\_frame\_id(ea):

def get\_frame\_lvar\_size(ea):

def get\_frame\_regs\_size(ea):

def get\_frame\_args\_size(ea):

def get\_frame\_size(ea):

def set\_frame\_size(ea, lvsize, frregs, argsize):

def get\_spd(ea):

def get\_sp\_delta(ea):

def add\_auto\_stkpnt(func\_ea, ea, delta):

def del\_stkpnt(func\_ea, ea):

def get\_min\_spd\_ea(func\_ea):

def get\_fixup\_target\_type(ea):

def get\_fixup\_target\_flags(ea):

def get\_fixup\_target\_sel(ea):

def get\_fixup\_target\_off(ea):

def get\_fixup\_target\_dis(ea):

def set\_fixup(ea, fixuptype, fixupflags, targetsel, targetoff, displ):

def get\_member\_qty(sid):

def get\_member\_id(sid, member\_offset):

def get\_prev\_offset(sid, offset):

def get\_next\_offset(sid, offset):

def get\_first\_member(sid):

def get\_last\_member(sid):

def get\_member\_offset(sid, member\_name):

def get\_member\_name(sid, member\_offset):

def get\_member\_cmt(sid, member\_offset, repeatable):

def get\_member\_size(sid, member\_offset):

def get\_member\_flag(sid, member\_offset):

def get\_member\_strid(sid, member\_offset):

def is\_union(sid):

def add\_struc(index, name, is\_union):

def del\_struc(sid):

def set\_struc\_idx(sid, index):

def add\_struc\_member(sid, name, offset, flag, typeid, nbytes, target=-1, tdelta=0, reftype=REF\_OFF32):

def del\_struc\_member(sid, member\_offset):

def set\_member\_name(sid, member\_offset, name):

def set\_member\_type(sid, member\_offset, flag, typeid, nitems, target=-1, tdelta=0, reftype=REF\_OFF32):

def set\_member\_cmt(sid, member\_offset, comment, repeatable):

def expand\_struc(sid, offset, delta, recalc):

def get\_fchunk\_attr(ea, attr):

def set\_fchunk\_attr(ea, attr, value):

def get\_next\_fchunk(ea):

def get\_prev\_fchunk(ea):

def append\_func\_tail(funcea, ea1, ea2):

def remove\_fchunk(funcea, tailea):

def set\_tail\_owner(tailea, funcea):

def first\_func\_chunk(funcea):

def next\_func\_chunk(funcea, tailea):

def get\_enum\_member(enum\_id, value, serial, bmask):

def get\_bmask\_name(enum\_id, bmask):

def get\_bmask\_cmt(enum\_id, bmask, repeatable):

def set\_bmask\_name(enum\_id, bmask, name):

def set\_bmask\_cmt(enum\_id, bmask, cmt, repeatable):

def get\_first\_enum\_member(enum\_id, bmask):

def get\_last\_enum\_member(enum\_id, bmask):

def get\_next\_enum\_member(enum\_id, value, bmask):

def get\_prev\_enum\_member(enum\_id, value, bmask):

def get\_enum\_member\_name(const\_id):

def get\_enum\_member\_cmt(const\_id, repeatable):

def add\_enum(idx, name, flag):

def add\_enum\_member(enum\_id, name, value, bmask):

def del\_enum\_member(enum\_id, value, serial, bmask):

def \_\_l2m1(v):

def rename(self, \*args): return 0

def kill(self, \*args): pass

def index(self, \*args): return -1

def altset(self, \*args): return 0

def supset(self, \*args): return 0

def altval(self, \*args): return 0

def supval(self, \*args): return 0

def altdel(self, \*args): return 0

def supdel(self, \*args): return 0

def altfirst(self, \*args): return -1

def supfirst(self, \*args): return -1

def altlast(self, \*args): return -1

def suplast(self, \*args): return -1

def altnext(self, \*args): return -1

def supnext(self, \*args): return -1

def altprev(self, \*args): return -1

def supprev(self, \*args): return -1

def hashset(self, \*args): return 0

def hashval(self, \*args): return 0

def hashstr(self, \*args): return 0

def hashstr\_buf(self, \*args): return 0

def hashset\_idx(self, \*args): return 0

def hashset\_buf(self, \*args): return 0

def hashval\_long(self, \*args): return 0

def hashdel(self, \*args): return 0

def hashfirst(self, \*args): return 0

def hashnext(self, \*args): return 0

def hashprev(self, \*args): return 0

def hashlast(self, \*args): return 0

def \_\_GetArrayById(array\_id):

def create\_array(name):

def get\_array\_id(name):

def rename\_array(array\_id, newname):

def delete\_array(array\_id):

def set\_array\_long(array\_id, idx, value):

def set\_array\_string(array\_id, idx, value):

def get\_array\_element(tag, array\_id, idx):

def del\_array\_element(tag, array\_id, idx):

def get\_first\_index(tag, array\_id):

def get\_last\_index(tag, array\_id):

def get\_next\_index(tag, array\_id, idx):

def get\_prev\_index(tag, array\_id, idx):

def set\_hash\_long(hash\_id, key, value):

def get\_hash\_long(hash\_id, key):

def set\_hash\_string(hash\_id, key, value):

def get\_hash\_string(hash\_id, key):

def del\_hash\_string(hash\_id, key):

def get\_first\_hash\_key(hash\_id):

def get\_last\_hash\_key(hash\_id):

def get\_next\_hash\_key(hash\_id, key):

def get\_prev\_hash\_key(hash\_id, key):

def add\_default\_til(name):

def import\_type(idx, type\_name):

def get\_type(ea):

def SizeOf(typestr):

def get\_tinfo(ea):

def get\_local\_tinfo(ordinal):

def guess\_type(ea):

def apply\_type(ea, py\_type, flags = TINFO\_DEFINITE):

def SetType(ea, newtype):

def parse\_decl(inputtype, flags):

def parse\_decls(inputtype, flags = 0):

def print\_decls(ordinals, flags):

def \_\_init\_\_(self):

def \_print(self, defstr):

def get\_ordinal\_qty():

def set\_local\_type(ordinal, input, flags):

def GetLocalType(ordinal, flags):

def get\_numbered\_type\_name(ordinal):

def update\_hidden\_range(ea, visible):

def \_get\_modules():

def get\_first\_module():

def get\_next\_module(base):

def get\_module\_name(base):

def get\_module\_size(base):

def resume\_process():

def send\_dbg\_command(cmd):

def get\_event\_id():

def get\_event\_pid():

def get\_event\_tid():

def get\_event\_ea():

def is\_event\_handled():

def get\_event\_module\_name():

def get\_event\_module\_base():

def get\_event\_module\_size():

def get\_event\_exit\_code():

def get\_event\_info():

def get\_event\_bpt\_hea():

def get\_event\_exc\_code():

def get\_event\_exc\_ea():

def can\_exc\_continue():

def get\_event\_exc\_info():

def set\_reg\_value(value, name):

def get\_bpt\_ea(n):

def get\_bpt\_attr(ea, bptattr):

def set\_bpt\_attr(address, bptattr, value):

def set\_bpt\_cond(ea, cnd, is\_lowcnd=0):

def enable\_tracing(trace\_level, enable):

def clear\_trace(filename):

def get\_color(ea, what):

def set\_color(ea, what, color):

def force\_bl\_jump(ea):

def force\_bl\_call(ea):

def set\_flag(off, bit, value):

def here(): return get\_screen\_ea()

def is\_mapped(ea): return (prev\_addr(ea+1)==ea)

E:\IDA\_Pro\_7.7\python\3>type idautils.py | find "def " >idautils.txt

def refs(ea, funcfirst, funcnext):

def CodeRefsTo(ea, flow):

def CodeRefsFrom(ea, flow):

def DataRefsTo(ea):

def DataRefsFrom(ea):

def XrefTypeName(typecode):

def XrefsFrom(ea, flags=0):

def XrefsTo(ea, flags=0):

def Threads():

def Heads(start=None, end=None):

def Functions(start=None, end=None):

def Chunks(start):

def Modules():

def Names():

def Segments():

def Entries():

def FuncItems(start):

def Structs():

def StructMembers(sid):

def DecodePrecedingInstruction(ea):

def DecodePreviousInstruction(ea):

def DecodeInstruction(ea):

def GetDataList(ea, count, itemsize=1):

def PutDataList(ea, datalist, itemsize=1):

def MapDataList(ea, length, func, wordsize=1):

def GetInputFileMD5():

def \_\_init\_\_(self, si):

def is\_1\_byte\_encoding(self):

def \_toseq(self, as\_unicode):

def \_\_str\_\_(self):

def \_\_unicode\_\_(self):

def clear\_cache(self):

def \_\_init\_\_(self, default\_setup = False):

def refresh(self):

def setup(self,

def \_get\_item(self, index):

def \_\_iter\_\_(self):

def \_\_getitem\_\_(self, index):

def GetIdbDir():

def GetRegisterList():

def GetInstructionList():

def \_Assemble(ea, line):

def Assemble(ea, line):

def \_copy\_obj(src, dest, skip\_list = None):

def \_\_init\_\_(self, reg, dtype):

def \_\_eq\_\_(self, other):

def \_\_getattr\_\_(self, attr):

def \_\_setattr\_\_(self, attr, value):

def \_\_getattr\_\_(self, name):

def \_\_setattr\_\_(self, name, value):

def \_\_init\_\_(self, actions, flags = 0):

def \_\_len\_\_(self):

def \_\_call\_\_(self):

def ProcessUiActions(actions, flags=0):

def \_\_init\_\_(self):

def \_\_str\_\_(self):

def header(self):

E:\IDA\_Pro\_7.7\python\3>type \*.py | find "def " >all.txt

E:\IDA\_Pro\_7.7\python\3>type all.txt | sort >all\_sort.txt