目录

IDA插件脚本：IDAPython

- 最新版本： `v0.8`
- 更新时间： `20240311`

简介

介绍IDA中支持用Python代码调用IDAPython接口编写插件脚本，实现各种功能。

源码+浏览+下载

本书的各种源码、在线浏览地址、多种格式文件下载如下：

HonKit源码

- crifan/ida_plugin_script_idapython: IDA插件脚本：IDAPython

如何使用此HonKit源码去生成发布为电子书

详见：crifan/honkit_template: demo how to use crifan honkit template and demo

在线浏览

- IDA插件脚本：IDAPython book.crifan.org
- IDA插件脚本：IDAPython crifan.github.io

离线下载阅读

- IDA插件脚本：IDAPython PDF
- IDA插件脚本：IDAPython ePub
- IDA插件脚本：IDAPython Mobi

版权和用途说明

鸣谢

感谢我的老婆**陈雪**的包容理解和悉心照料，才使得我 `crifan` 有更多精力去专注技术专研和整理归纳出这些电子书和技术教程，特此鸣谢。

其他

作者的其他电子书

本人 `crifan` 还写了其他 `150+` 本电子书教程，感兴趣可移步至：

crifan/crifan_ebook_readme: Crifan的电子书的使用说明

关于作者

关于作者更多介绍，详见：

关于CrifanLi李茂 – 在路上

IDA插件脚本概述

IDA中，支持通过，写Python代码，调用 `IDAPython` 的API接口，实现各种功能。对应的Python脚本，叫做：

- IDA插件==IDA脚本=IDA的Python脚本

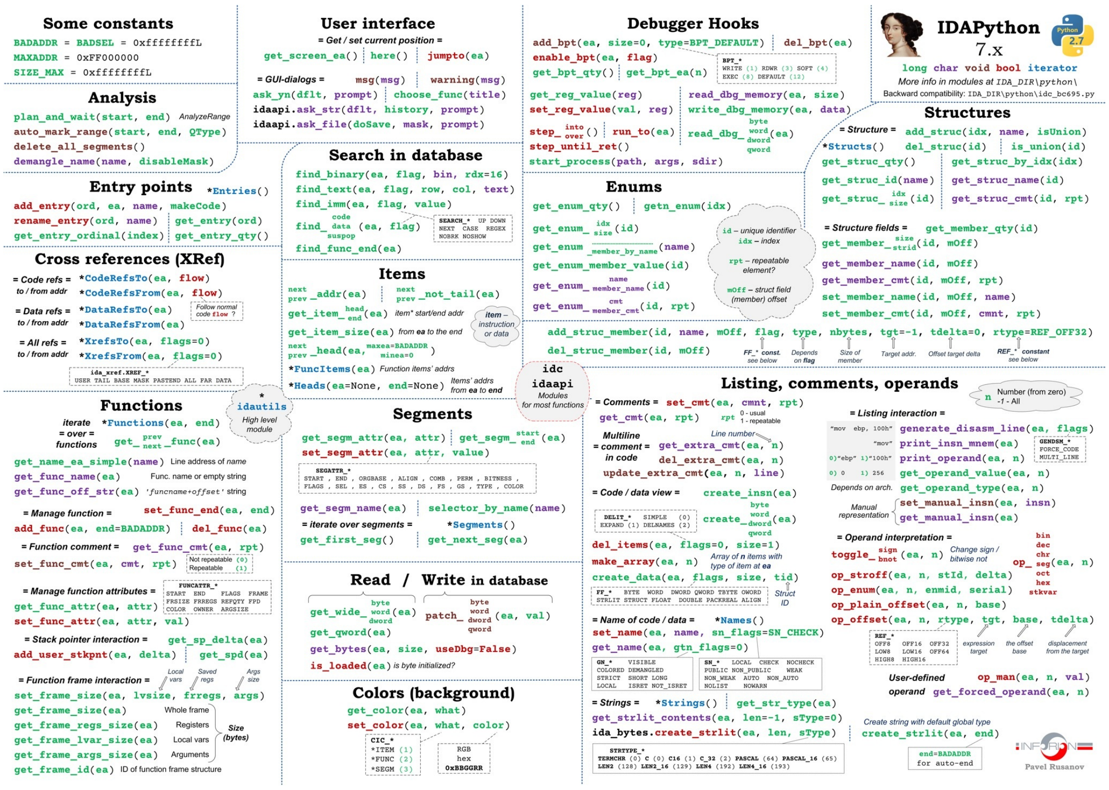
IDAPython相关文档和资料

- `IDAPython` 文档
  - 入口
    - IDAPython documentation
  - 常用（子）模块
    - idaapi
      - https://hex-rays.com/products/ida/support/idapython_docs/ida_idaapi.html
- `IDAPython` 资料
  - Github
    - https://github.com/idapython
      - src源码
        - idapython/src: IDAPython project for Hex-Ray's IDA Pro

IDAPython的cheatsheet小抄

有别人整理好的，IDA的IDAPython的 `cheatsheet` = 小抄

- IDAPython cheatsheet
  - IDAPython 7.x



IDAPython 7.x
long char void bool iterator
More info in modules at `IDA_DIR\python\`
Backward compatibility: `IDA_DIR\python\idc_bc695.py`

Some constants
```
BADADDR = BADSEL = 0xffffffffL
MAXADDR = 0xFF000000
SIZE_MAX = 0xffffffffL
```

Analysis
```
plan_and_wait(start, end)  AnalyzeRange
auto_mark_range(start, end, QType)
delete_all_segments()
demangle_name(name, disableMask)
```

Entry points `*Entries()`
```
add_entry(ord, ea, name, makeCode)
rename_entry(ord, name)     | get_entry(ord)
get_entry_ordinal(index)    | get_entry_qty()
```

Cross references (XRef)
```
= Code refs =       *CodeRefsTo(ea, flow)
to / from addr      *CodeRefsFrom(ea, flow)     Follow normal code flow ?
= Data refs =       *DataRefsTo(ea)
to / from addr      *DataRefsFrom(ea)
= All refs =        *XrefsTo(ea, flags=0)
to / from addr      *XrefsFrom(ea, flags=0)
    ida_xref.XREF_*
    USER TAIL BASE MASK PASTEND ALL FAR DATA
```

* idautils — High level module

Functions
```
iterate
= over =       *Functions(ea, end)
functions      get_ prev next _func(ea)
get_name_ea_simple(name)   Line address of name
get_func_name(ea)          Func. name or empty string
get_func_off_str(ea)       'funcname+offset' string
= Manage function =        set_func_end(ea, end)
add_func(ea, end=BADADDR) | del_func(ea)
= Function comment =       get_func_cmt(ea, rpt)
set_func_cmt(ea, cmt, rpt)    Not repeatable (0)
                              Repeatable     (1)
= Manage function attributes =
get_func_attr(ea, attr)       FUNCATTR_*
set_func_attr(ea, attr, val)  START  END    FLAGS  FRAME
                              FRSIZE FRREGS REFQTY FPD
                              COLOR  OWNER  ARGSIZE
= Stack pointer interaction =  get_sp_delta(ea)
add_user_stkpnt(ea, delta) | get_spd(ea)
= Function frame interaction =
set_frame_size(ea, lvsize, frregs, args)
                   Local vars  Saved regs  Args size
get_frame_size(ea)          Whole frame
get_frame_regs_size(ea)     Registers      Size
get_frame_lvar_size(ea)     Local vars     (bytes)
get_frame_args_size(ea)     Arguments
get_frame_id(ea)   ID of function frame structure
```

User interface
```
= Get / set current position =
get_screen_ea() | here() | jumpto(ea)
= GUI-dialogs =   msg(msg) | warning(msg)
ask_yn(dflt, prompt) | choose_func(title)
idaapi.ask_str(dflt, history, prompt)
idaapi.ask_file(doSave, mask, prompt)
```

Search in database
```
find_binary(ea, flag, bin, rdx=16)
find_text(ea, flag, row, col, text)
find_imm(ea, flag, value)
find_ code data suspop (ea, flag)      SEARCH_*  UP DOWN
find_func_end(ea)                      NEXT  CASE  REGEX
                                       NOBRK NOSHOW
```

Items
```
next prev _addr(ea) | next prev _not_tail(ea)
get_item_ head end (ea)   item* start/end addr
get_item_size(ea)   from ea to the end           item – instruction or data
next prev _head(ea, maxea=BADADDR minea=0)
*FuncItems(ea)   Function items' addrs
*Heads(ea=None, end=None)   Items' addrs from ea to end
```

idc idaapi — Modules for most functions

Segments
```
get_segm_attr(ea, attr) | get_segm_ start end (ea)
set_segm_attr(ea, attr, value)
    SEGATTR_*
    START , END , ORGBASE , ALIGN , COMB , PERM , BITNESS ,
    FLAGS , SEL , ES , CS , SS , DS , FS , GS , TYPE , COLOR
get_segm_name(ea) | selector_by_name(name)
= iterate over segments =   *Segments()
get_first_seg() | get_next_seg(ea)
```

Read / Write in database
```
get_wide_ byte word dword (ea)     patch_ byte word dword qword (ea, val)
get_qword(ea)
get_bytes(ea, size, useDbg=False)
is_loaded(ea)  is byte initialized?
```

Colors (background)
```
get_color(ea, what)
set_color(ea, what, color)
  CIC_*        RGB
  *ITEM (1)    hex
  *FUNC (2)    0xBBGGRR
  *SEGM (3)
```

Debugger Hooks
```
add_bpt(ea, size=0, type=BPT_DEFAULT) | del_bpt(ea)
enable_bpt(ea, flag)                  BPT_*
get_bpt_qty() | get_bpt_ea(n)         WRITE (1) RDWR (3) SOFT (4)
                                      EXEC (8) DEFAULT (12)
get_reg_value(reg)          | read_dbg_memory(ea, size)
set_reg_value(val, reg)     | write_dbg_memory(ea, data)
step_ into over () | run_to(ea) | read_dbg_ byte word dword qword (ea)
step_until_ret()
start_process(path, args, sdir)
```

Enums
```
get_enum_qty() | getn_enum(idx)
get_enum_ idx size (id)
get_enum _member_by_name (name)
get_enum_member_value(id)
get_enum_ name member_name (id)
get_enum_ cmt member_cmt (id, rpt)
```
id – unique identifier
idx – index
rpt – repeatable element?
mOff – struct field (member) offset

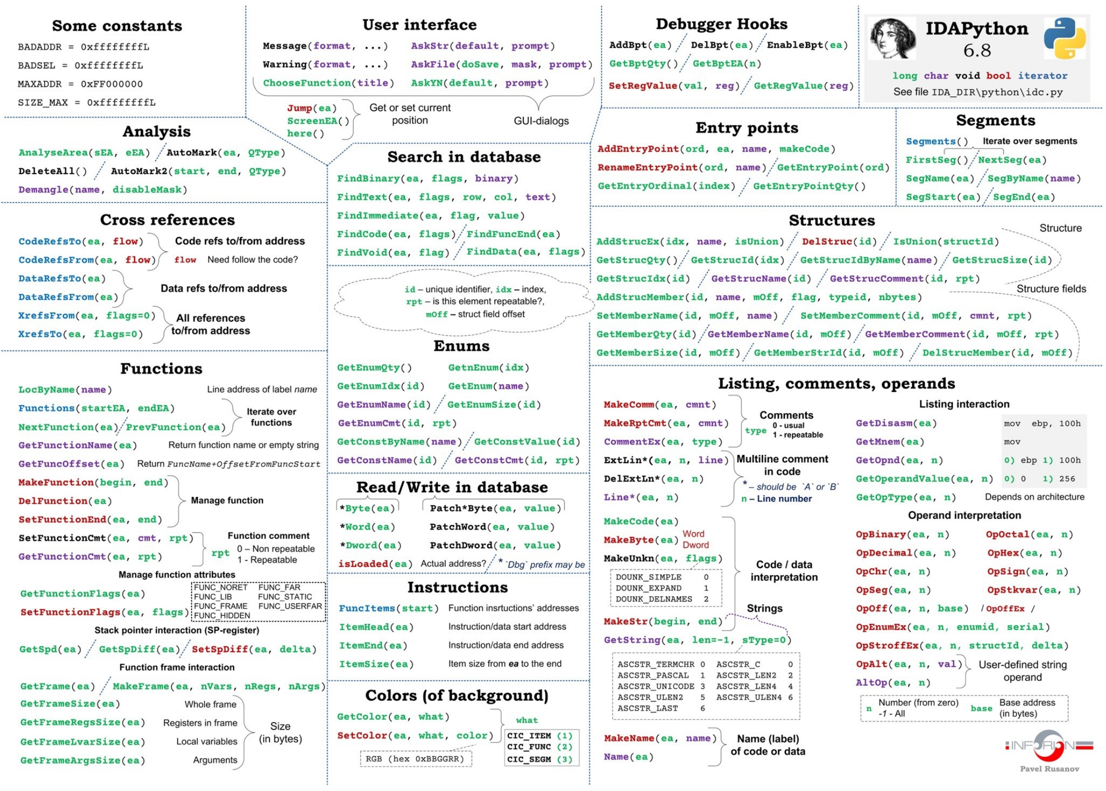
Structures
```
= Structure =   add_struc(idx, name, isUnion)
*Structs()      del_struc(id) | is_union(id)
get_struc_qty()          | get_struc_by_idx(idx)
get_struc_id(name)       | get_struc_name(id)
get_struc_ idx size (id) | get_struc_cmt(id, rpt)
= Structure fields =   get_member_qty(id)
get_member_ size strid (id, mOff)
get_member_name(id, mOff)
get_member_cmt(id, mOff, rpt)
set_member_name(id, mOff, name)
set_member_cmt(id, mOff, cmnt, rpt)
add_struc_member(id, name, mOff, flag, type, nbytes, tgt=-1, tdelta=0, rtype=REF_OFF32)
del_struc_member(id, mOff)
   flag: FF_* const. see below; type: Depends on flag; nbytes: Size of member;
   tgt: Target addr.; tdelta: Offset target delta; rtype: REF_* constant see below
```

Listing, comments, operands   n — Number (from zero), -1 - All
```
= Comments =   set_cmt(ea, cmnt, rpt)
get_cmt(ea, rpt)       rpt  0 - usual
                            1 - repeatable
Multiline
= comment =   get_extra_cmt(ea, n)      (n: Line number)
in code       del_extra_cmt(ea, n)
update_extra_cmt(ea, n, line)
= Code / data view =   create_insn(ea)
   DELIT_*  SIMPLE   (0)     create_ byte word dword qword (ea)
   EXPAND (1) DELNAMES (2)
del_items(ea, flags=0, size=1)
make_array(ea, n)    Array of n items with type of item at ea
create_data(ea, flags, size, tid)      tid: Struct ID
   FF_*   BYTE  WORD  DWORD QWORD TBYTE OWORD
   STRLIT STRUCT FLOAT DOUBLE PACKREAL ALIGN
= Name of code / data =   *Names()
set_name(ea, name, sn_flags=SN_CHECK)
get_name(ea, gtn_flags=0)
   GN_*     VISIBLE              SN_*   LOCAL  CHECK  NOCHECK
   COLORED  DEMANGLED            PUBLIC NON_PUBLIC    WEAK
   STRICT   SHORT LONG           NON_WEAK  AUTO  NON_AUTO
   LOCAL    ISRET NOT_ISRET      NOLIST    NOWARN
= Strings =   *Strings() | get_str_type(ea)
get_strlit_contents(ea, len=-1, sType=0)
ida_bytes.create_strlit(ea, len, sType) | create_strlit(ea, end)
                                          Create string with default global type
                                          end=BADADDR for auto-end
   STRTYPE_*
   TERMCHR (0) C (0) C16 (1) C_32 (2) PASCAL (64) PASCAL_16 (65)
   LEN2 (128) LEN2_16 (129) LEN4 (192) LEN4_16 (193)

= Listing interaction =
"mov  ebp, 100h"     generate_disasm_line(ea, flags)    GENDSM_*
"mov"                print_insn_mnem(ea)                FORCE_CODE
0)"ebp" 1)"100h"     print_operand(ea, n)               MULTI_LINE
0) 0    1) 256       get_operand_value(ea, n)
Depends on arch.     get_operand_type(ea, n)
Manual               set_manual_insn(ea, insn)
representation       get_manual_insn(ea)
= Operand interpretation =
toggle_ sign bnot (ea, n)   Change sign / bitwise not
op_ bin dec chr seg oct hex stkvar (ea, n)
op_stroff(ea, n, stId, delta)
op_enum(ea, n, enmid, serial)
op_plain_offset(ea, n, base)
op_offset(ea, n, rtype, tgt, base, tdelta)
   REF_*                     tgt: expression target
   OFF8  OFF16  OFF32        base: the offset base
   LOW8  LOW16  OFF64        tdelta: displacement from the target
   HIGH8 HIGH16
User-defined   op_man(ea, n, val)
operand        get_forced_operand(ea, n)
```

INFORION
Pavel Rusanov

  - IDAPython 6.x



IDAPython 6.8
long char void bool iterator
See file `IDA_DIR\python\idc.py`

Some constants
```
BADADDR = 0xffffffffL
BADSEL = 0xffffffffL
MAXADDR = 0xFF000000
SIZE_MAX = 0xffffffffL
```

User interface
```
Message(format, ...)     AskStr(default, prompt)
Warning(format, ...)     AskFile(doSave, mask, prompt)
ChooseFunction(title)    AskYN(default, prompt)
                                             GUI-dialogs
Jump(ea)
ScreenEA()    Get or set current position
here()
```

Analysis
```
AnalyseArea(sEA, eEA) / AutoMark(ea, QType)
DeleteAll() / AutoMark2(start, end, QType)
Demangle(name, disableMask)
```

Cross references
```
CodeRefsTo(ea, flow)       Code refs to/from address
CodeRefsFrom(ea, flow)     flow  Need follow the code?
DataRefsTo(ea)             Data refs to/from address
DataRefsFrom(ea)
XrefsFrom(ea, flags=0)     All references
XrefsTo(ea, flags=0)       to/from address
```

Functions
```
LocByName(name)                        Line address of label name
Functions(startEA, endEA)              Iterate over
NextFunction(ea)/PrevFunction(ea)      functions
GetFunctionName(ea)                    Return function name or empty string
GetFuncOffset(ea)                      Return FuncName+OffsetFromFuncStart
MakeFunction(begin, end)
DelFunction(ea)                        Manage function
SetFunctionEnd(ea, end)
SetFunctionCmt(ea, cmt, rpt)           Function comment
GetFunctionCmt(ea, rpt)                rpt  0 – Non repeatable
                                            1 - Repeatable
Manage function attributes
GetFunctionFlags(ea)                   FUNC_NORET   FUNC_FAR
SetFunctionFlags(ea, flags)            FUNC_LIB     FUNC_STATIC
                                       FUNC_FRAME   FUNC_USERFAR
                                       FUNC_HIDDEN
Stack pointer interaction (SP-register)
GetSpd(ea) / GetSpDiff(ea)/ SetSpDiff(ea, delta)
Function frame interaction
GetFrame(ea) / MakeFrame(ea, nVars, nRegs, nArgs)
GetFrameSize(ea)          Whole frame
GetFrameRegsSize(ea)      Registers in frame     Size
GetFrameLvarSize(ea)      Local variables        (in bytes)
GetFrameArgsSize(ea)      Arguments
```

Search in database
```
FindBinary(ea, flags, binary)
FindText(ea, flags, row, col, text)
FindImmediate(ea, flag, value)
FindCode(ea, flags)/ FindFuncEnd(ea)
FindVoid(ea, flag) / FindData(ea, flags)
```
id – unique identifier, idx – index,
rpt – is this element repeatable?,
mOff – struct field offset

Enums
```
GetEnumQty()            GetnEnum(idx)
GetEnumIdx(id)          GetEnum(name)
GetEnumName(id) / GetEnumSize(id)
GetEnumCmt(id, rpt)
GetConstByName(name) / GetConstValue(id)
GetConstName(id) / GetConstCmt(id, rpt)
```

Read/Write in database
```
*Byte(ea)       Patch*Byte(ea, value)
*Word(ea)       PatchWord(ea, value)
*Dword(ea)      PatchDword(ea, value)
isLoaded(ea)  Actual address? / * `Dbg` prefix may be
```

Instructions
```
FuncItems(start)    Function insrtuctions' addresses
ItemHead(ea)        Instruction/data start address
ItemEnd(ea)         Instruction/data end address
ItemSize(ea)        Item size from ea to the end
```

Colors (of background)
```
GetColor(ea, what)              what
SetColor(ea, what, color)       CIC_ITEM (1)
RGB (hex 0xBBGGRR)              CIC_FUNC (2)
                                CIC_SEGM (3)
```

Debugger Hooks
```
AddBpt(ea) / DelBpt(ea) / EnableBpt(ea)
GetBptQty() / GetBptEA(n)
SetRegValue(val, reg) / GetRegValue(reg)
```

Entry points
```
AddEntryPoint(ord, ea, name, makeCode)
RenameEntryPoint(ord, name) / GetEntryPoint(ord)
GetEntryOrdinal(index) / GetEntryPointQty()
```

Segments
```
Segments()   Iterate over segments
FirstSeg() / NextSeg(ea)
SegName(ea) / SegByName(name)
SegStart(ea) / SegEnd(ea)
```

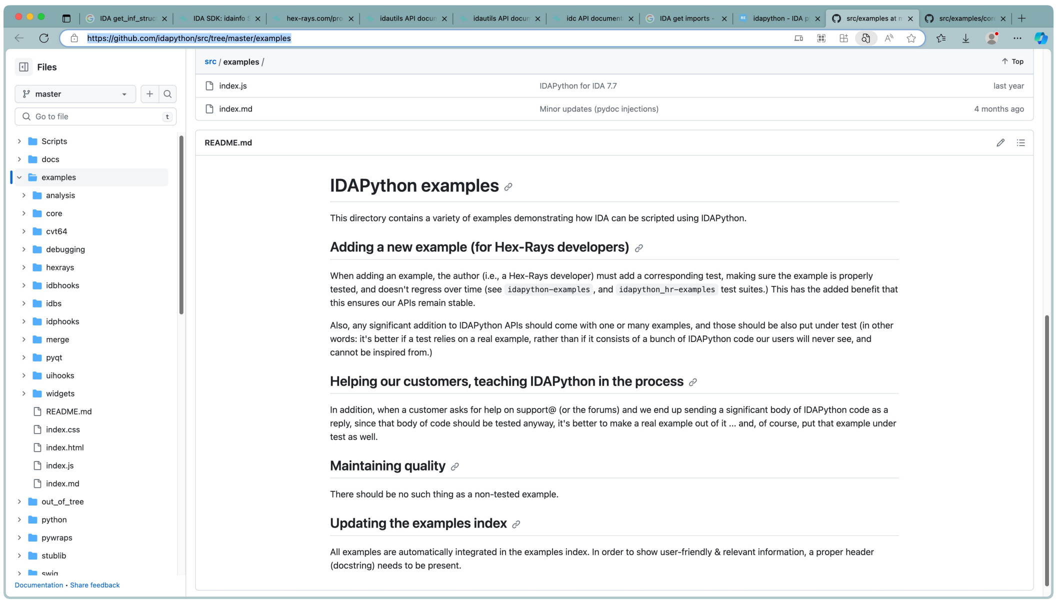
Structures
```
AddStrucEx(idx, name, isUnion) / DelStruc(id) / IsUnion(structId)          Structure
GetStrucQty() / GetStrucId(idx) / GetStrucIdByName(name) / GetStrucSize(id)
GetStrucIdx(id) / GetStrucName(id) / GetStrucComment(id, rpt)
AddStrucMember(id, name, mOff, flag, typeid, nbytes)                       Structure fields
SetMemberName(id, mOff, name) / SetMemberComment(id, mOff, cmnt, rpt)
GetMemberQty(id) / GetMemberName(id, mOff) / GetMemberComment(id, mOff, rpt)
GetMemberSize(id, mOff) / GetMemberStrId(id, mOff) / DelStrucMember(id, mOff)
```

Listing, comments, operands
```
MakeComm(ea, cmnt)          Comments
MakeRptCmt(ea, cmnt)        type  0 - usual
CommentEx(ea, type)               1 - repeatable
ExtLin*(ea, n, line)        Multiline comment
DelExtLn*(ea, n)            in code
Line*(ea, n)                * – should be `A` or `B`
                            n – Line number
MakeCode(ea)
MakeByte(ea)  Word Dword
MakeUnkn(ea, flags)         Code / data
  DOUNK_SIMPLE     0        interpretation
  DOUNK_EXPAND     1
  DOUNK_DELNAMES   2
MakeStr(begin, end)         Strings
GetString(ea, len=-1, sType=0)
  ASCSTR_TERMCHR 0   ASCSTR_C      0
  ASCSTR_PASCAL  1   ASCSTR_LEN2   2
  ASCSTR_UNICODE 3   ASCSTR_LEN4   4
  ASCSTR_ULEN2   5   ASCSTR_ULEN4  6
  ASCSTR_LAST    6
MakeName(ea, name)          Name (label)
Name(ea)                    of code or data

Listing interaction
GetDisasm(ea)               mov  ebp, 100h
GetMnem(ea)                 mov
GetOpnd(ea, n)              0) ebp 1) 100h
GetOperandValue(ea, n)      0) 0   1) 256
GetOpType(ea, n)            Depends on architecture
Operand interpretation
OpBinary(ea, n)       OpOctal(ea, n)
OpDecimal(ea, n)      OpHex(ea, n)
OpChr(ea, n)          OpSign(ea, n)
OpSeg(ea, n)          OpStkvar(ea, n)
OpOff(ea, n, base)  / OpOffEx /
OpEnumEx(ea, n, enumid, serial)
OpStroffEx(ea, n, structId, delta)
OpAlt(ea, n, val)     User-defined string
AltOp(ea, n)          operand
n  Number (from zero)      base  Base address
   -1 - All                      (in bytes)
```

INFORION
Pavel Rusanov

IDA官网示例代码

- IDA官网示例代码
  - 入口
    - src/examples at master · idapython/src (github.com)
    - 

https://github.com/idapython/src/tree/master/examples

Files
master
Go to file

Scripts
docs
examples
analysis
core
cvt64
debugging
hexrays
idbhooks
idbs
idphooks
merge
pyqt
uihooks
widgets
README.md
index.css
index.html
index.js
index.md
out_of_tree
python
pywraps
stublib
swig

Documentation · Share feedback

src / examples /

index.js	IDAPython for IDA 7.7	last year
index.md	Minor updates (pydoc injections)	4 months ago

README.md

IDAPython examples

This directory contains a variety of examples demonstrating how IDA can be scripted using IDAPython.

Adding a new example (for Hex-Rays developers)

When adding an example, the author (i.e., a Hex-Rays developer) must add a corresponding test, making sure the example is properly tested, and doesn't regress over time (see `idapython-examples`, and `idapython_hr-examples` test suites.) This has the added benefit that this ensures our APIs remain stable.

Also, any significant addition to IDAPython APIs should come with one or many examples, and those should be also put under test (in other words: it's better if a test relies on a real example, rather than if it consists of a bunch of IDAPython code our users will never see, and cannot be inspired from.)

Helping our customers, teaching IDAPython in the process

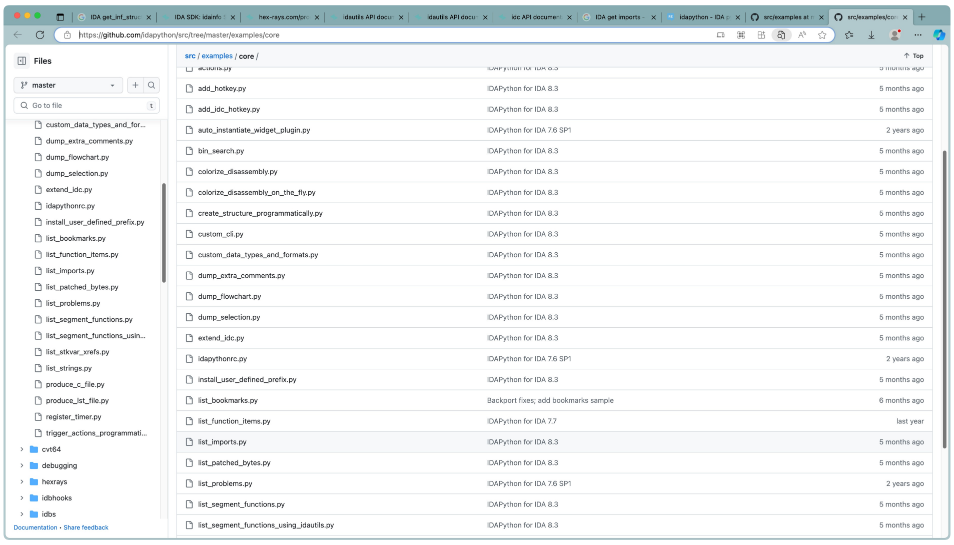
In addition, when a customer asks for help on support@ (or the forums) and we end up sending a significant body of IDAPython code as a reply, since that body of code should be tested anyway, it's better to make a real example out of it ... and, of course, put that example under test as well.

Maintaining quality

There should be no such thing as a non-tested example.

Updating the examples index

All examples are automatically integrated in the examples index. In order to show user-friendly & relevant information, a proper header (docstring) needs to be present.

  - 包含
    - core核心
      - src/examples/core at master · idapython/src (github.com)
        - 图
          - 

https://github.com/idapython/src/tree/master/examples/core

Files
master
Go to file

custom_data_types_and_for...
dump_extra_comments.py
dump_flowchart.py
dump_selection.py
extend_idc.py
idapythonrc.py
install_user_defined_prefix.py
list_bookmarks.py
list_function_items.py
list_imports.py
list_patched_bytes.py
list_problems.py
list_segment_functions.py
list_segment_functions_usin...
list_stkvar_xrefs.py
list_strings.py
produce_c_file.py
produce_lst_file.py
register_timer.py
trigger_actions_programmati...
cvt64
debugging
hexrays
idbhooks
idbs

Documentation · Share feedback

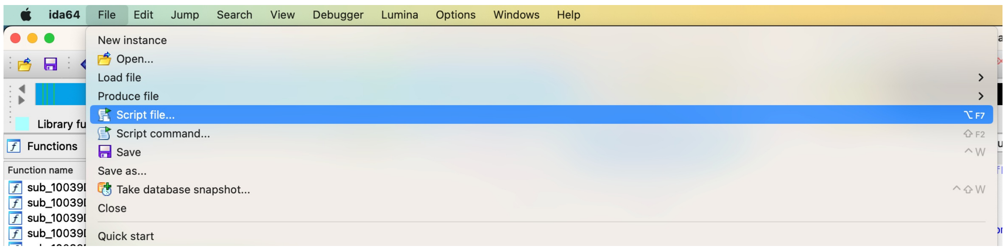
src / examples / core /

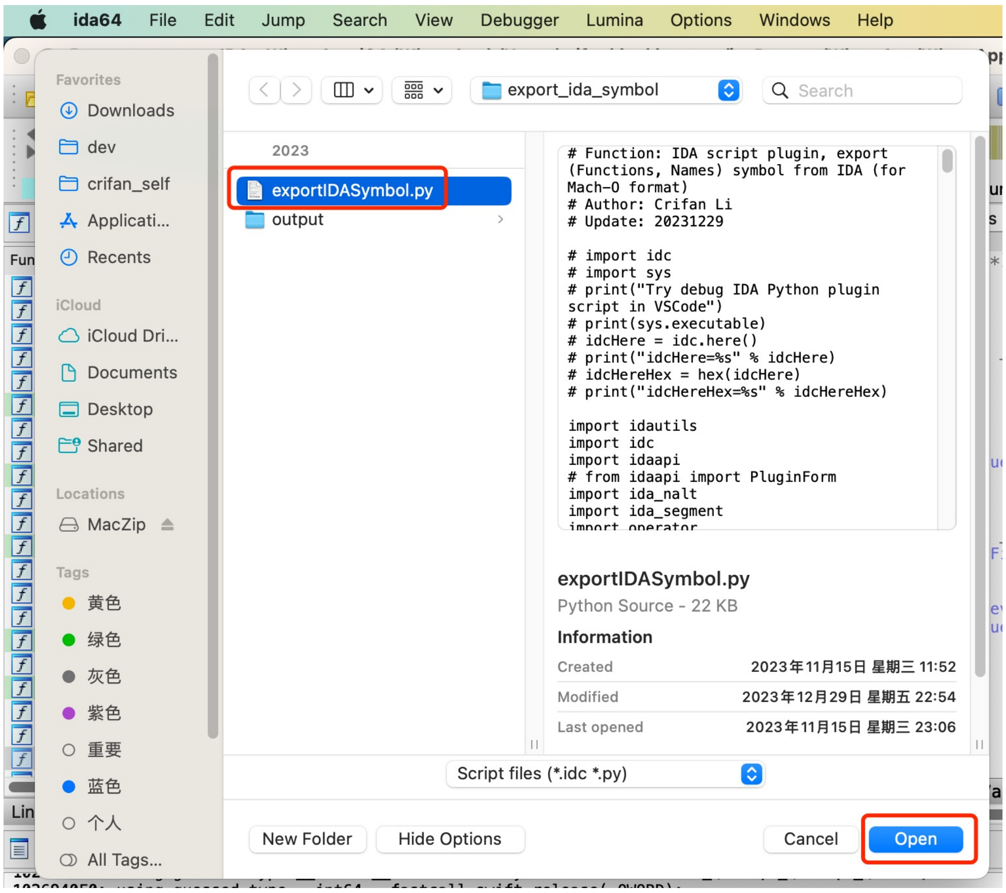
actions.py	IDAPython for IDA 8.3	5 months ago
add_hotkey.py	IDAPython for IDA 8.3	5 months ago
add_idc_hotkey.py	IDAPython for IDA 8.3	5 months ago
auto_instantiate_widget_plugin.py	IDAPython for IDA 7.6 SP1	2 years ago
bin_search.py	IDAPython for IDA 8.3	5 months ago
colorize_disassembly.py	IDAPython for IDA 8.3	5 months ago
colorize_disassembly_on_the_fly.py	IDAPython for IDA 8.3	5 months ago
create_structure_programmatically.py	IDAPython for IDA 8.3	5 months ago
custom_cli.py	IDAPython for IDA 8.3	5 months ago
custom_data_types_and_formats.py	IDAPython for IDA 8.3	5 months ago
dump_extra_comments.py	IDAPython for IDA 8.3	5 months ago
dump_flowchart.py	IDAPython for IDA 8.3	5 months ago
dump_selection.py	IDAPython for IDA 8.3	5 months ago
extend_idc.py	IDAPython for IDA 8.3	5 months ago
idapythonrc.py	IDAPython for IDA 7.6 SP1	2 years ago
install_user_defined_prefix.py	IDAPython for IDA 8.3	5 months ago
list_bookmarks.py	Backport fixes; add bookmarks sample	6 months ago
list_function_items.py	IDAPython for IDA 7.7	last year
list_imports.py	IDAPython for IDA 8.3	5 months ago
list_patched_bytes.py	IDAPython for IDA 8.3	5 months ago
list_problems.py	IDAPython for IDA 7.6 SP1	2 years ago
list_segment_functions.py	IDAPython for IDA 8.3	5 months ago
list_segment_functions_using_idautils.py	IDAPython for IDA 8.3	5 months ago

        - 包括
          - list_bookmarks.py
          - list_function_items.py
          - list_imports.py
          - list_segment_functions.py
          - list_strings.py
          - 等等

2024-03-11 09:42:33

运行IDA脚本插件

- 运行IDA脚本插件
  - 最典型方式：加载脚本文件= `Script file`
    - 步骤
      - `IDA` -> `File` -> `Script file` ->选择对应IDA的Python脚本文件-》即可运行
    - 图
      - Mac
        - 

        - 

```
# Function: IDA script plugin, export
(Functions, Names) symbol from IDA (for
Mach-O format)
# Author: Crifan Li
# Update: 20231229

# import idc
# import sys
# print("Try debug IDA Python plugin
script in VSCode")
# print(sys.executable)
# idcHere = idc.here()
# print("idcHere=%s" % idcHere)
# idcHereHex = hex(idcHere)
# print("idcHereHex=%s" % idcHereHex)

import idautils
import idc
import idaapi
# from idaapi import PluginForm
import ida_nalt
import ida_segment
import operator
```

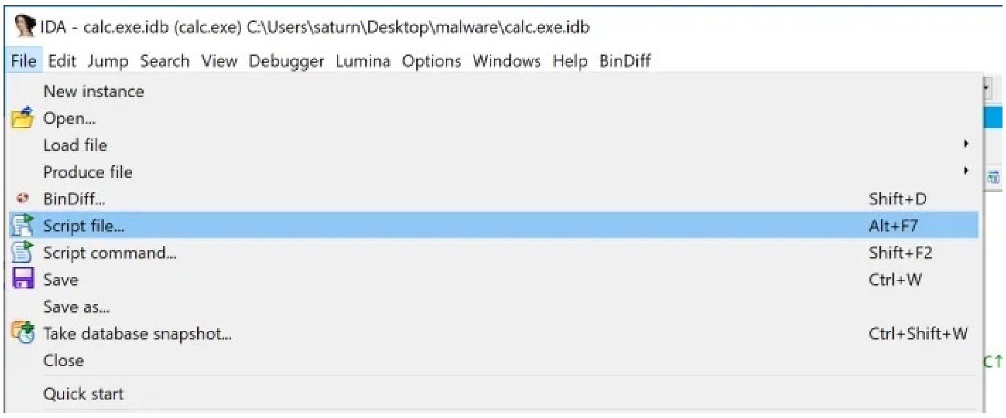
exportIDASymbol.py
Python Source - 22 KB

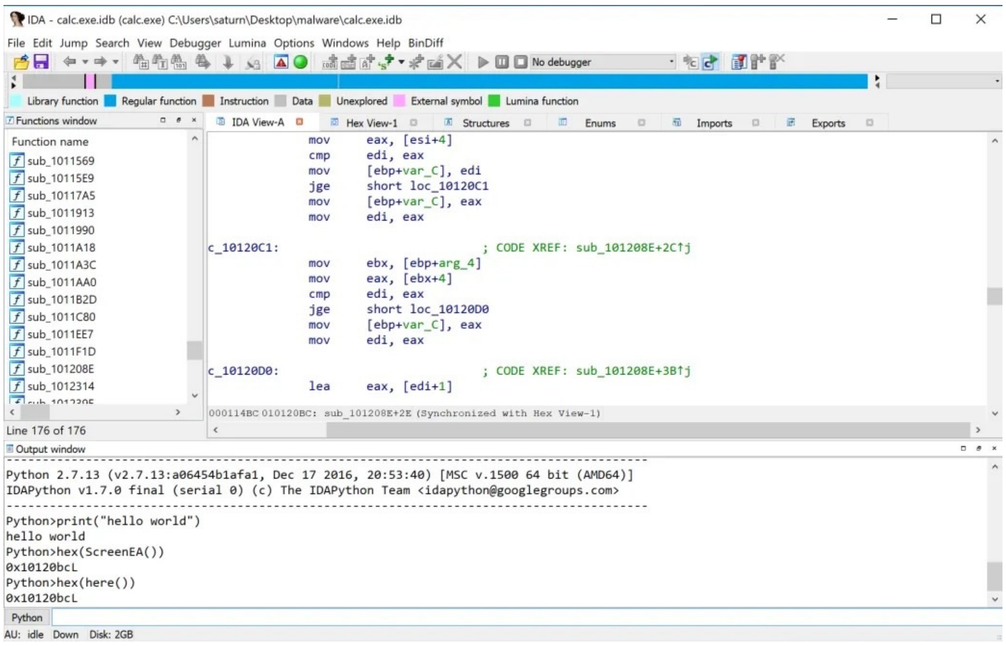
Information

Created	2023年11月15日 星期三 11:52
Modified	2023年12月29日 星期五 22:54
Last opened	2023年11月15日 星期三 23:06

Script files (*.idc *.py)

      - Win

- 
IDA - calc.exe.idb (calc.exe) C:\Users\saturn\Desktop\malware\calc.exe.idb
File Edit Jump Search View Debugger Lumina Options Windows Help BinDiff
New instance
Open...
Load file
Produce file
BinDiff... Shift+D
Script file... Alt+F7
Script command... Shift+F2
Save Ctrl+W
Save as...
Take database snapshot... Ctrl+Shift+W
Close
Quick start

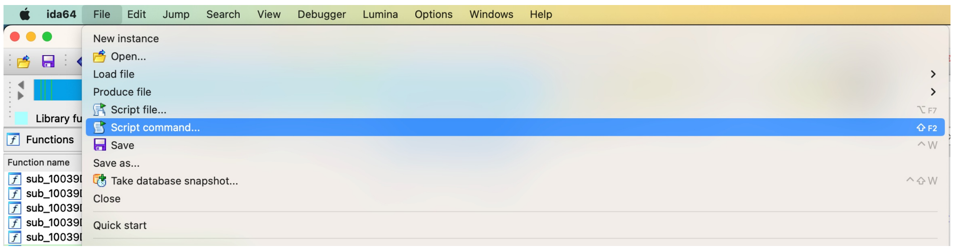
- 其他方式
  - `IDAPython Interpreter` = `IDAPython交互式命令行解析器`
    - 主要用途：临时写点Python脚本代码测试
    - 界面
      - 

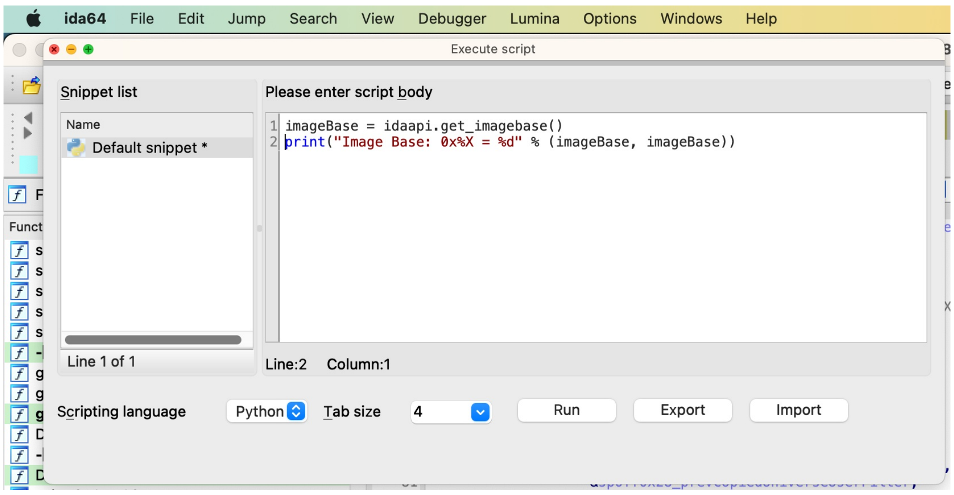
```
                mov     eax, [esi+4]
                cmp     edi, eax
                mov     [ebp+var_C], edi
                jge     short loc_10120C1
                mov     [ebp+var_C], eax
                mov     edi, eax

c_10120C1:                              ; CODE XREF: sub_101208E+2C↑j
                mov     ebx, [ebp+arg_4]
                mov     eax, [ebx+4]
                cmp     edi, eax
                jge     short loc_10120D0
                mov     [ebp+var_C], eax
                mov     edi, eax

c_10120D0:                              ; CODE XREF: sub_101208E+3B↑j
                lea     eax, [edi+1]
000114BC 010120BC: sub_101208E+2E (Synchronized with Hex View-1)
```

```
---------------------------------------------------------------------------------------
Python 2.7.13 (v2.7.13:a06454b1afa1, Dec 17 2016, 20:53:40) [MSC v.1500 64 bit (AMD64)]
IDAPython v1.7.0 final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>
---------------------------------------------------------------------------------------
Python>print("hello world")
hello world
Python>hex(ScreenEA())
0x10120bcL
Python>hex(here())
0x10120bcL
```

  - `Script Command`
    - 步骤
      - `IDA` -> `File` -> `Script Command` ->自己输入要运行的Python脚本
    - 图
      - Mac
        - 

- 

```
imageBase = idaapi.get_imagebase()
print("Image Base: 0x%X = %d" % (imageBase, imageBase))
```

调试

常用函数

IDA函数

- 概述
    - 关于IDA的Python脚本代码中常用的函数，已整理至
        - crifanIDA.py
    - 使用举例
        - AutoRename.py
        - exportIDASymbol.py
        - ida_search_block.py
- 详解

辅助内容

导入IDA的库

```
import re
import os

import idc
import idaapi
import idautils
import ida_nalt
import ida_segment
```

辅助函数

- crifaniOS.py

isObjcFunctionName

```
def isObjcFunctionName(funcName):
  """
  check is ObjC function name or not
  eg:
    "+[WAAvatarStringsActions editAvatar]" -> True
    "-[ParentGroupInfoViewController initWithParentGroupChatSession:userContext:recentl
yLinkedGroupJIDs:]" -> True
    "-[OKEvolveSegmentationVC proCard]_116" -> True
    "-[WAAvatarStickerUpSellSupplementaryView .cxx_destruct]" -> True
    "sub_10004C6D8" -> False
    "protocol witness for RawRepresentable.init(rawValue:) in conformance UIFont.FontWe
ight" -> True
  """
  isMatchObjcFuncName = re.match("^[\-\+]\[\w+ [\w\.\:]+\]\w*$", funcName)
  isObjcFuncName = bool(isMatchObjcFuncName)
  # print("funcName=%s -> isObjcFuncName=%s" % (funcName, isObjcFuncName))
  return isObjcFuncName
```

IDA常用函数

ida_getInfo

```
def ida_getInfo():
  """
  get IDA info
  """
  info = idaapi.get_inf_structure()
  # print("info=%s" % info)
  return info
```

ida_printInfo

```
def ida_printInfo(info):
  """
  print IDA info
  """
  version = info.version
  print("version=%s" % version)
  is64Bit = info.is_64bit()
  print("is64Bit=%s" % is64Bit)
  procName = info.procname
  print("procName=%s" % procName)
  entryPoint = info.start_ea
  print("entryPoint=0x%X" % entryPoint)
  baseAddr = info.baseaddr
  print("baseAddr=0x%X" % baseAddr)
```

ida_printAllImports

```
def ida_printAllImports():
  """
  print all imports lib and functions inside lib"""
  nimps = ida_nalt.get_import_module_qty()
  print("Found %d import(s)..." % nimps)
  for i in range(nimps):
    name = ida_nalt.get_import_module_name(i)
    if not name:
      print("Failed to get import module name for [%d] %s" % (i, name))
      name = "<unnamed>"
    else:
      print("[%d] %s" % (i, name))

    def imp_cb(ea, name, ordinal):
        if not name:
            print("%08x: ordinal #%d" % (ea, ordinal))
        else:
            print("%08x: %s (ordinal #%d)" % (ea, name, ordinal))
        # True -> Continue enumeration
        # False -> Stop enumeration
        return True
    ida_nalt.enum_import_names(i, imp_cb)
```

ida_printSegment

```
def ida_printSegment(curSeg):
  """
  print segment info
    Note: in IDA, segment == section
  """
  segName = curSeg.name
  # print("type(segName)=%s" % type(segName))
  segSelector = curSeg.sel
  segStartAddr = curSeg.start_ea
  segEndAddr = curSeg.end_ea
  print("Segment: [0x%X-0x%X] name=%s, selector=%s : seg=%s" % (segStartAddr, segEndAddr
, segName, segSelector, curSeg))
```

ida_getSegmentList

```
def ida_getSegmentList():
  """
  get segment list
  """
  segList = []
  segNum = ida_segment.get_segm_qty()
  for segIdx in range(segNum):
    curSeg = ida_segment.getnseg(segIdx)
    # print("curSeg=%s" % curSeg)
    segList.append(curSeg)
    # ida_printSegment(curSeg)
  return segList
```

ida_testGetSegment

```
def ida_testGetSegment():
  """
  test get segment info
  """
  # textSeg = ida_segment.get_segm_by_name("__TEXT")
  # dataSeg = ida_segment.get_segm_by_name("__DATA")

  # ida_getSegmentList()

  # NAME___TEXT = "21"
  # NAME___TEXT = 21
  # NAME___TEXT = "__TEXT,__text"
  # NAME___TEXT = "__TEXT:__text"
  # NAME___TEXT = ".text"
```

```
"""
  __TEXT,__text
  __TEXT,__stubs
  __TEXT,__stub_helper
  __TEXT,__objc_stubs
  __TEXT,__const
  __TEXT,__objc_methname
  __TEXT,__cstring
  __TEXT,__swift5_typeref
  __TEXT,__swift5_protos
  __TEXT,__swift5_proto
  __TEXT,__swift5_types
  __TEXT,__objc_classname
  __TEXT,__objc_methtype
  __TEXT,__gcc_except_tab
  __TEXT,__ustring
  __TEXT,__unwind_info
  __TEXT,__eh_frame
  __TEXT,__oslogstring

  __DATA,__got
  __DATA,__la_symbol_ptr
  __DATA,__mod_init_func
  __DATA,__const
  __DATA,__cfstring
  __DATA,__objc_classlist
  __DATA,__objc_catlist
  __DATA,__objc_protolist
  __DATA,__objc_imageinfo
  __DATA,__objc_const
  __DATA,__objc_selrefs
  __DATA,__objc_protorefs
  __DATA,__objc_classrefs
  __DATA,__objc_superrefs
  __DATA,__objc_ivar
  __DATA,__objc_data
  __DATA,__data
  __DATA,__objc_stublist
  __DATA,__swift_hooks
  __DATA,__swift51_hooks
  __DATA,__s_async_hook
  __DATA,__swift56_hooks
  __DATA,__thread_vars
  __DATA,__thread_bss
  __DATA,__bss
  __DATA,__common
"""

# __TEXT,__text
NAME___text = "__text"
textSeg = ida_segment.get_segm_by_name(NAME___text)
print("textSeg: %s -> %s" % (NAME___text, textSeg))
ida_printSegment(textSeg)

# __TEXT,__objc_methname
NAME___objc_methname = "__objc_methname"
```

```
  objcMethNameSeg = ida_segment.get_segm_by_name(NAME___objc_methname)
  print("objcMethNameSeg: %s -> %s" % (NAME___objc_methname, objcMethNameSeg))
  ida_printSegment(objcMethNameSeg)

  # __DATA,__got
  NAME___got = "__got"
  gotSeg = ida_segment.get_segm_by_name(NAME___got)
  print("gotSeg: %s -> %s" % (NAME___got, gotSeg))
  ida_printSegment(gotSeg)

  # __DATA,__data
  # NAME___DATA = "22"
  # NAME___DATA = 22
  NAME___DATA = "__data"
  dataSeg = ida_segment.get_segm_by_name(NAME___DATA)
  print("dataSeg: %s -> %s" % (NAME___DATA, dataSeg))
  ida_printSegment(dataSeg)

  # exist two one: __TEXT,__const / __DATA,__const
  NAME___const = "__const"
  constSeg = ida_segment.get_segm_by_name(NAME___const)
  print("constSeg: %s -> %s" % (NAME___const, constSeg))
  ida_printSegment(constSeg)
```

ida_getDemangledName

```
def ida_getDemangledName(origSymbolName):
  """
  use IDA to get demangled name for original symbol name
  """
  retName = origSymbolName
  # demangledName = idc.demangle_name(origSymbolName, idc.get_inf_attr(idc.INF_SHORT_DN
))
  # https://hex-rays.com/products/ida/support/ida74_idapython_no_bc695_porting_guide.sh
tml
  demangledName = idc.demangle_name(origSymbolName, idc.get_inf_attr(idc.INF_SHORT_DEMN
AMES))
  if demangledName:
    retName = demangledName

  # do extra post process:
  # remove/replace invalid char for non-objc function name
  isNotObjcFuncName = not isObjcFunctionName(retName)
  # print("isNotObjcFuncName=%s" % isNotObjcFuncName)
  if isNotObjcFuncName:
    retName = retName.replace("?", "")
    retName = retName.replace(" ", "_")
    retName = retName.replace("*", "_")
  # print("origSymbolName=%s -> retName=%s" % (origSymbolName, retName))
  return retName
```

ida_getFunctionEndAddr

```
def ida_getFunctionEndAddr(funcAddr):
  """
  get function end address
    Example:
      0x1023A2534 -> 0x1023A2540
  """
  funcAddrEnd = idc.get_func_attr(funcAddr, attr=idc.FUNCATTR_END)
  return funcAddrEnd
```

ida_getFunctionSize

```
def ida_getFunctionSize(funcAddr):
  """
  get function size
    Example:
      0x1023A2534 -> 12
  """
  funcAddrEnd = idc.get_func_attr(funcAddr, attr=idc.FUNCATTR_END)
  funcAddStart = idc.get_func_attr(funcAddr, attr=idc.FUNCATTR_START)
  funcSize = funcAddrEnd - funcAddStart
  return funcSize
```

ida_getFunctionName

```
def ida_getFunctionName(funcAddr):
  """
  get function name
    Exmaple:
      0x1023A2534 -> "sub_1023A2534"
      0xF9D260 -> "objc_msgSend$initWithKeyValueStore_namespace_binaryCoders_X22toX0_X2
3toX2_X24toX4_EF8C"
  """
  funcName = idc.get_func_name(funcAddr)
  return funcName
```

ida_getName

```
def ida_getName(curAddr):
  """
  get name
    Exmaple:
      0xF9D260 -> "_objc_msgSend$initWithKeyValueStore:namespace:binaryCoders:"
  """
  addrName = idc.get_name(curAddr)
  return addrName
```

ida_getDisasmStr

```
def ida_getDisasmStr(funcAddr):
  """
  get disasmemble string
    Exmaple:
      0x1023A2534 -> "MOV X5, X0"
  """
  # method 1: generate_disasm_line
  # disasmLine_forceCode = idc.generate_disasm_line(funcAddr, idc.GENDSM_FORCE_CODE)
  # print("disasmLine_forceCode: type=%s, val=%s" % (type(disasmLine_forceCode), disasm
Line_forceCode))
  # disasmLine_multiLine = idc.generate_disasm_line(funcAddr, idc.GENDSM_MULTI_LINE)
  # print("disasmLine_multiLine: type=%s, val=%s" % (type(disasmLine_multiLine), disasm
Line_multiLine))

  # method 2: GetDisasm
  disasmLine = idc.GetDisasm(funcAddr)
  # print("disasmLine: type=%s, val=%s" % (type(disasmLine), disasmLine))

  # post process
  # print("disasmLine=%s" % disasmLine)
  # "MOV              X4, X21" -> "MOV X4, X21"
  disasmLine = re.sub("\s+", " ", disasmLine)
  # print("disasmLine=%s" % disasmLine)
  return disasmLine
```

ida_getFunctionAddrList

```
def ida_getFunctionAddrList():
  """
  get function address list
  """
  functionIterator = idautils.Functions()
  functionAddrList = []
  for curFuncAddr in functionIterator:
    functionAddrList.append(curFuncAddr)
  return functionAddrList
```

ida_rename

```
def ida_rename(curAddr, newName, retryName=None):
  """
  rename <curAddr> to <newName>. if fail, retry with with <retryName> if not None
    Example:
      0x3B4E28, "X2toX21_X1toX20_X0toX19_4E28", "X2toX21_X1toX20_X0toX19_3B4E28" -> Tru
e, "X2toX21_X1toX20_X0toX19_4E28"
  """
  # print("curAddr=0x%X, newName=%s, retryName=%s" % (curAddr, newName, retryName))
  isRenameOk = False
  renamedName = None

  isOk = idc.set_name(curAddr, newName)
  # print("isOk=%s for [0x%X] -> %s" % (isOk, curAddr, newName))
  if isOk == 1:
    isRenameOk = True
    renamedName = newName
  else:
    if retryName:
      isOk = idc.set_name(curAddr, retryName)
      # print("isOk=%s for [0x%X] -> %s" % (isOk, curAddr, retryName))
      if isOk == 1:
        isRenameOk = True
        renamedName = retryName

  # print("isRenameOk=%s, renamedName=%s" % (isRenameOk, renamedName))
  return (isRenameOk, renamedName)
```

ida_getCurrentFolder

```
def ida_getCurrentFolder():
  """
  get current folder for IDA current opened binary file
    Example:
      -> /Users/crifan/dev/dev_root/iosReverse/WhatsApp/ipa/Payload/WhatsApp.app
      -> /Users/crifan/dev/dev_root/iosReverse/WhatsApp/ipa/Payload/WhatsApp.app/Framew
orks/SharedModules.framework
  """
  curFolder = None
  inputFileFullPath = ida_nalt.get_input_file_path()
  # print("inputFileFullPath=%s" % inputFileFullPath)
  if inputFileFullPath.startswith("/var/containers/Bundle/Application"):
    # inputFileFullPath=/var/containers/Bundle/Application/2BE964D4-8DF0-4858-A06D-66CA
8741ACDC/WhatsApp.app/WhatsApp
    # -> maybe IDA bug -> after debug settings, output iOS device path, but later no au
thority to write exported file to it
    # so need to avoid this case, change to output to PC side (Mac) current folder
    curFolder = "."
  else:
    curFolder = os.path.dirname(inputFileFullPath)
  # print("curFolder=%s" % curFolder)

  # debugInputPath = ida_nalt.dbg_get_input_path()
  # print("debugInputPath=%s" % debugInputPath)

  curFolder = os.path.abspath(curFolder)
  # print("curFolder=%s" % curFolder)
  # here work:
  # . -> /Users/crifan/dev/dev_root/iosReverse/WhatsApp/ipa/Payload/WhatsApp.app
  return curFolder
```

isDefaultTypeForObjcMsgSendFunction

```
def isDefaultTypeForObjcMsgSendFunction(funcAddr):
  """
  check is objc_msgSend$xxx function's default type "id(void *, const char *, ...)" or
```

```
not
  eg:
    0xF3EF8C -> True
      note: funcType=id(void *, const char *, __int64, __int64, ...)
  """
  isDefType = False
  funcType = idc.get_type(funcAddr)
  # print("[0x%X] -> funcType=%s" % (funcAddr, funcType))
  if funcType:
    defaultTypeMatch = re.search("\.\.\.\)$", funcType)
    # print("defaultTypeMatch=%s" % defaultTypeMatch)
    isDefType = bool(defaultTypeMatch)
    # print("isDefType=%s" % isDefType)
  return isDefType
```

无需调用IDA的API的相关函数

isDefaultSubFuncName

```
def isDefaultSubFuncName(funcName):
  """
  check is default sub_XXX function or not from name
  eg:
    sub_F332C0 -> True, "F332C0"
  """
  isSub = False
  addressStr = None
  # subMatch = re.match("^sub_[0-9A-Za-z]+$", funcName)
  subMatch = re.match("^sub_(?P<addressStr>[0-9A-Fa-f]+)$", funcName)
  # print("subMatch=%s" % subMatch)
  if subMatch:
    isSub = True
    addressStr = subMatch.group("addressStr")
  return isSub, addressStr
```

isReservedPrefix_loc

```
def isReservedPrefix_loc(funcName):
  """
  check is reserved prefix loc_XXX name or not
  eg:
    loc_100007A2C -> True, "100007A2C"
  """
  isLoc = False
  addressStr = None
  locMatch = re.match("^loc_(?P<addressStr>[0-9A-Fa-f]+)$", funcName)
  # print("locMatch=%s" % locMatch)
  if locMatch:
    isLoc = True
    addressStr = locMatch.group("addressStr")
  return isLoc, addressStr
```

isDefaultSubFunction

```
def isDefaultSubFunction(curAddr):
  """
  check is default sub_XXX function or not from address
  """
  isDefSubFunc = False
  curFuncName  = ida_getFunctionName(curAddr)
  # print("curFuncName=%s" % curFuncName)
  if curFuncName:
    isDefSubFunc, subAddStr = isDefaultSubFuncName(curFuncName)
  return isDefSubFunc, curFuncName
```

isObjcMsgSendFunction

```
def isObjcMsgSendFunction(curAddr):
  """
  check is default sub_XXX function or not from address
  """
  isObjcMsgSend = False
  curFuncName  = ida_getFunctionName(curAddr)
  # print("curFuncName=%s" % curFuncName)
  if curFuncName:
    isObjcMsgSend, selectorStr = isObjcMsgSendFuncName(curFuncName)
  return isObjcMsgSend, selectorStr
```

IDA工具类

Operand=操作数

```
################################################################################
# Config & Settings & Const
################################################################################

ArmSpecialRegNameList = [
  "SB",
  "TR",
  "XR",
  "IP",
  "IP0",
  "IP1",
  "PR",
  "SP",
  "FP",
  "LR",
  "PC",
]

class Operand:
  # Operand Type
  # https://hex-rays.com/products/ida/support/idapython_docs/idc.html#idc.get_operand_v
alue
  o_void      = 0       # No Operand                           ----------
  o_reg       = 1       # General Register (al,ax,es,ds...)    reg
  o_mem       = 2       # Direct Memory Reference  (DATA)      addr
  o_phrase    = 3       # Memory Ref [Base Reg + Index Reg]    phrase
  o_displ     = 4       # Memory Reg [Base Reg + Index Reg + Displacement] phrase+addr
  o_imm       = 5       # Immediate Value                      value
  o_far       = 6       # Immediate Far Address  (CODE)        addr
  o_near      = 7       # Immediate Near Address (CODE)        addr
  o_idpspec0  = 8       # Processor specific type
  o_idpspec1  = 9       # Processor specific type
  o_idpspec2  = 10      # Processor specific type
  o_idpspec3  = 11      # Processor specific type
  o_idpspec4  = 12      # Processor specific type
  o_idpspec5  = 13      # Processor specific type
                        # There can be more processor specific types

  # x86
  o_trreg  =       o_idpspec0      # trace register
  o_dbreg  =       o_idpspec1      # debug register
  o_crreg  =       o_idpspec2      # control register
  o_fpreg  =       o_idpspec3      # floating point register
  o_mmxreg  =      o_idpspec4      # mmx register
  o_xmmreg  =      o_idpspec5      # xmm register

  # arm
  o_reglist  =     o_idpspec1      # Register list (for LDM/STM)
  o_creglist   =     o_idpspec2      # Coprocessor register list (for CDP)
  o_creg  =          o_idpspec3      # Coprocessor register (for LDC/STC)
  o_fpreglist  =     o_idpspec4      # Floating point register list
  o_text  =          o_idpspec5      # Arbitrary text stored in the operand
  o_cond  =          o_idpspec5 + 1  # ARM condition as an operand

  # ppc
  o_spr  =           o_idpspec0      # Special purpose register
  o_twofpr  =        o_idpspec1      # Two FPRs
  o_shmbme  =        o_idpspec2      # SH & MB & ME
  o_crf  =           o_idpspec3      # crfield      x.reg
  o_crb  =           o_idpspec4      # crbit        x.reg
  o_dcr  =           o_idpspec5      # Device control register


  # addStr = "add"
  # addStr = "Add"
  offStr = "Off" # Offset=Index
  # valStr = "val"
  valStr = "Val"

  def __init__(self, operand, type, value):
    self.operand = operand
    self.type = type
    self.value = value

    # for o_displ / o_phrase
    self.baseReg = None
    self.indexReg = None
    # for o_displ
    self.displacement = None

    self._postInit()

  def _postInit(self):
    # print("_postInit")
    if self.isDispl():
      # o_displ     = 4        # Memory Reg [Base Reg + Index Reg + Displacement] phrase
+addr
      # [SP,#arg_18]
      # [X20,#0x50]
      # print("self.operand=%s" % self.operand)
      # displMatch = re.search("\[(?P<baseReg>\w+),(?P<displacement>#[\w\-\.]+)\]", sel
f.operand)
      # [X9]
      displMatch = re.search("\[(?P<baseReg>\w+)(,(?P<displacement>#[\w\-\.]+))?\]", se
lf.operand)
      # print("displMatch=%s" % displMatch)
      if displMatch:
        self.baseReg = displMatch.group("baseReg")
        # print("self.baseReg=%s" % self.baseReg)
        self.displacement = displMatch.group("displacement")
        # print("self.displacement=%s" % self.displacement)
    elif self.isPhrase():
      # o_phrase    = 3        # Memory Ref [Base Reg + Index Reg]    phrase
      # [X19,X8]
```

```
        # print("self.operand=%s" % self.operand)
        phraseMatch = re.search("\[(?P<baseReg>\w+),(?P<indexReg>\w+)\]", self.operand)
        # print("phraseMatch=%s" % phraseMatch)
        if phraseMatch:
          self.baseReg = phraseMatch.group("baseReg")
          # print("self.baseReg=%s" % self.baseReg)
          self.indexReg = phraseMatch.group("indexReg")
          # print("self.indexReg=%s" % self.indexReg)

  def __str__(self):
    valStr = ""
    if self.value <= 0:
      valStr = "%s" % self.value
    else:
      valStr = "0x%X" % self.value
    # curOpStr = "<Operand: op=%s,type=%d,val=%s>" % (self.operand, self.type, valStr)
    # curOpStr = "<Operand: op=%s,type=%d,val=%s, baseReg=%s,indexReg=%s,displ=%s>" % (
self.operand, self.type, valStr, self.baseReg, self.indexReg, self.displacement)
    extraInfo = ""
    if self.isDispl():
      extraInfo = ",bsReg=%s,idxReg=%s,displ=%s" % (self.baseReg, self.indexReg, self.d
isplacement)
    elif self.isPhrase():
      extraInfo = ",bsReg=%s,idxReg=%s" % (self.baseReg, self.indexReg)
    curOpStr = "<Operand: op=%s,type=%d,val=%s%s>" % (self.operand, self.type, valStr,
extraInfo)
    # print("curOpStr=%s" % curOpStr)
    return curOpStr

  @staticmethod
  def listToStr(operandList):
    # operandStrList = []
    # for curOperand in operandList:
    #   if curOperand:
    #     curOperandStr = "%s" % curOperand
    #   else:
    #     curOperandStr = ""
    #   # print("curOperandStr=%s" % curOperandStr)
    #   operandStrList.append(curOperandStr)
    operandStrList = [str(eachOperand) for eachOperand in operandList]
    operandListAllStr = ", ".join(operandStrList)
    operandListAllStr = "[%s]" % operandListAllStr
    return operandListAllStr

  def isReg(self):
    return self.type == Operand.o_reg

  def isImm(self):
    return self.type == Operand.o_imm

  def isDispl(self):
    return self.type == Operand.o_displ

  def isPhrase(self):
    return self.type == Operand.o_phrase
  def isNear(self):
    return self.type == Operand.o_near

  def isIdpspec0(self):
    #   o_idpspec0 = 8        # Processor specific type
    return self.type == Operand.o_idpspec0

  def isValid(self):
    isDebug = False

    # isValidOperand = bool(self.operand)
    # print("isValidOperand=%s" % isValidOperand)
    # if isValidOperand:
    isValidOperand = False

    if isDebug:
      print("self.operand=%s" % self.operand)

    if self.operand:
      if self.isImm():
        # #0x20200A2C
        # #0x2020
        # #arg_20
        # isMatchImm = re.match("^#[0-9a-fA-FxX]+$", self.operand)
        # #-3.0
        # isMatchImm = re.match("^#\w+$", self.operand)
        isMatchImm = re.match("^#[\w\-\.]+$", self.operand)
        if isDebug:
          print("isMatchImm=%s" % isMatchImm)
        isValidOperand = bool(isMatchImm)
        if isDebug:
          print("isValidOperand=%s" % isValidOperand)
      elif self.isReg():
        # X0/X1
        # D8/D4
        # Special: XZR/WZR
        regNameUpper = self.operand.upper()
        # print("regNameUpper=%s" % regNameUpper)
        # isMatchReg = re.match("^[XD]\d+$", regNameUpper)
        # isMatchReg = re.match("^[XDW]\d+$", regNameUpper)
        isMatchReg = re.match("^([XDW]\d+)|(XZR)|(WZR)$", regNameUpper)
        if isDebug:
          print("isMatchReg=%s" % isMatchReg)
        isValidOperand = bool(isMatchReg)
        if isDebug:
          print("isValidOperand=%s" % isValidOperand)
        if not isValidOperand:
          isValidOperand = regNameUpper in ArmSpecialRegNameList
      elif self.isDispl():
        # o_displ    = 4        # Memory Reg [Base Reg + Index Reg + Displacement] phra
se+addr
        # curOperand=<Operand: op=[SP,#arg_18],type=4,val=0x18>
        # if self.baseReg and (not self.indexReg) and self.displacement:
        # curOperand=<Operand: op=[X9],type=4,val=0x0>
        if isDebug:
          print("self.baseReg=%s, self.indexReg=%s, self.displacement=%s" % (self.baseR
eg, self.indexReg, self.displacement))

        if self.baseReg and (not self.indexReg):
          # Note: self.displacement is None / Not-None
          # TODO: add more type support, like indexReg not None
          isValidOperand = True
      elif self.isPhrase():
        # curOperand=<Operand: op=[X19,X8],type=3,val=0x94>
        if isDebug:
          print("self.baseReg=%s, self.indexReg=%s" % (self.baseReg, self.indexReg))
        if self.baseReg and self.indexReg:
          isValidOperand = True
      elif self.isNear():
        # o_near     = 7        # Immediate Near Address (CODE)        addr
        # curOperand=<Operand: op=_objc_copyWeak,type=7,val=0x1024ABBD0>
        if isDebug:
          print("self.value=%s" % self.value)

        if self.value:
          # jump to some (non 0) address -> consider is valid
          isValidOperand = True
      elif self.isIdpspec0():
        isValidOperand = True

    # print("isValidOperand=%s" % isValidOperand)

    # isValidType = self.type != Operand.o_void
    # isValidValue = self.value >= 0
    # isValidAll = isValidOperand and isValidType and isValidValue
    # isValidTypeValue = False
    # if self.isReg() or self.isImm():
    #   isValidTypeValue = self.value >= 0
    # elif self.isIdpspec0():
    #   isValidTypeValue = self.value == -1

    if self.isIdpspec0():
      isValidTypeValue = self.value == -1
    else:
      isValidType = self.type != Operand.o_void
      isValidValue = self.value >= 0
      isValidTypeValue = isValidType and isValidValue
    isValidAll = isValidOperand and isValidTypeValue

    if isDebug:
      print("Operand isValidAll=%s" % isValidAll)
    return isValidAll

  def isInvalid(self):
    return not self.isValid()

  @property
  def immVal(self):
    curImmVal = None
    if self.isImm():
      curImmVal = self.value
      # print("curImmVal=%s" % curImmVal)
      return curImmVal

    @property
    def immValHex(self):
      curImmValHex = ""
      if self.immVal != None:
        curImmValHex = "0x%X" % self.immVal
        # print("curImmValHex=%s" % curImmValHex)
      return curImmValHex

    @property
    def regName(self):
      curRegName = None
      if self.isReg():
        curRegName = self.operand
      return curRegName

    @property
    def contentStr(self):
      contentStr = ""
      if self.isReg():
        # print("isReg")
        contentStr = self.regName
      elif self.isImm():
        # print("isImm")
        # if 0 == self.immVal:
        # for 0 <= x < 8, not add 0x prefix, eg: 0x7 -> 7
        if (self.immVal >= 0) and (self.immVal < 8):
          # contentStr = "0"
          contentStr = "%X" % self.immVal
        else:
          contentStr = self.immValHex
      elif self.isIdpspec0():
          contentStr = self.operand
      elif self.isDispl():
          # [SP,#arg_18]
          # print("self.displacement=%s" % self.displacement)
          if self.displacement:
            displacementStr = ""
            if self.value != None:
              if (self.value >= 0) and (self.value < 8):
                displacementStr = "%X" % self.value
              else:
                displacementStr = "0x%X" % self.value
            # print("displacementStr=%s" % displacementStr)
            contentStr = "%s%s%s%s" % (self.baseReg, Operand.offStr, displacementStr, Ope
rand.valStr)
          else:
            contentStr = "%s%s" % (self.baseReg, Operand.valStr)
      elif self.isPhrase():
        # [X19,X8]
        contentStr = "%s%s%s%s" % (self.baseReg, Operand.offStr, self.indexReg, Operand.v
alStr)

      # remove invalid char
      # <Operand: op=W0,UXTB,type=8,val=-1>
    # W0,UXTB -> W0UXTB
    contentStr = contentStr.replace(",", "")
    # X21,LSL#32
    # X8,ASR#29
    contentStr = contentStr.replace("#", "")

    # TODO: add more case

    # print("contentStr=%s" % contentStr)
    return contentStr

  @property
  def regIdx(self):
    curRegIdx = None
    if self.isReg():
      # TODO: extract reg idx,
      # eg: X0 -> 0, X4 -> 4
      # note: additonal: D0 -> 0, D8 -> 8 ?
      curRegIdx = 0
    return curRegIdx
```

Instruction=指令

```
# class Instruction(object):
class Instruction:
  # toStr = "to"
  toStr = "To"
  # addStr = "add"
  addStr = "Add"

  def __init__(self, addr, name, operands):
    self.addr = addr
    self.disAsmStr = ida_getDisasmStr(addr)
    # print("self.disAsmStr=%s" % self.disAsmStr)
    self.name = name
    self.operands = operands

  def __str__(self):
    # operandsAllStr = Operand.listToStr(self.operands)
    # print("operandsAllStr=%s" % operandsAllStr)
    # curInstStr = "<Instruction: addr=0x%X,name=%s,operands=%s>" % (self.addr, self.na
me, operandsAllStr)
    # curInstStr = "<Instruction: addr=0x%X,disAsmStr=%s>" % (self.addr, self.disAsmStr)

    curInstStr = "<Instruction: 0x%X: %s>" % (self.addr, self.disAsmStr)
    # print("curInstStr=%s" % curInstStr)
    return curInstStr

  @staticmethod
  def listToStr(instList):
    instContentStrList = [str(eachInst) for eachInst in instList]
    instListAllStr = ", ".join(instContentStrList)
    instListAllStr = "[%s]" % instListAllStr
    return instListAllStr
```

```
@staticmethod
def parse(addr):
  isDebug = False
  # # if addr == 0x10235D610:
  # # if addr == 0x1002B8340:
  # if addr == 0x102390B18:
  #   isDebug = True
  # isDebug = True

  if isDebug:
    print("Instruction: parsing 0x%X" % addr)
  parsedInst = None

  instName = idc.print_insn_mnem(addr)
  if isDebug:
    print("instName=%s" % instName)

  curOperandIdx = 0
  curOperandVaild = True
  operandList = []
  while curOperandVaild:
    if isDebug:
      logSubSub("[%d]" % curOperandIdx)
    curOperand = idc.print_operand(addr, curOperandIdx)
    if isDebug:
      print("curOperand=%s" % curOperand)
    curOperandType = idc.get_operand_type(addr, curOperandIdx)
    if isDebug:
      print("curOperandType=%d" % curOperandType)
    curOperandValue = idc.get_operand_value(addr, curOperandIdx)
    if isDebug:
      print("curOperandValue=%s=0x%X" % (curOperandValue, curOperandValue))
    curOperand = Operand(curOperand, curOperandType, curOperandValue)
    if isDebug:
      print("curOperand=%s" % curOperand)
    if curOperand.isValid():
      operandList.append(curOperand)
    else:
      if isDebug:
        print("End of operand for invalid %s" % curOperand)
      curOperandVaild = False

    if isDebug:
      print("curOperandVaild=%s" % curOperandVaild)
    curOperandIdx += 1

  if operandList:
    parsedInst = Instruction(addr=addr, name=instName, operands=operandList)
  if isDebug:
    print("parsedInst=%s" % parsedInst)
    print("operandList=%s" % Operand.listToStr(operandList))
  return parsedInst

def isInst(self, instName):
  isMatchInst = False
```

```
        if self.name:
          if (instName.lower() == self.name.lower()):
            isMatchInst = True
        return isMatchInst

    @property
    def contentStr(self):
        """
        convert to meaningful string of Instruction real action / content
        """
        contentStr = ""

        isDebug = False
        # isDebug = True

        if isDebug:
          print("self=%s" % self)

        operandNum = len(self.operands)
        if isDebug:
          print("operandNum=%s" % operandNum)

        isPairInst = self.isStp() or self.isLdp()
        if isDebug:
          print("isPairInst=%s" % isPairInst)
        if not isPairInst:
          if operandNum >= 2:
            srcOperand = self.operands[1]
            if isDebug:
              print("srcOperand=%s" % srcOperand)
            srcOperandStr = srcOperand.contentStr
            if isDebug:
              print("srcOperandStr=%s" % srcOperandStr)
            dstOperand = self.operands[0]
            if isDebug:
              print("dstOperand=%s" % dstOperand)
            dstOperandStr = dstOperand.contentStr
            if isDebug:
              print("dstOperandStr=%s" % dstOperandStr)

        if self.isMov() or self.isFmov():
          # MOV X0, X24
          # FMOV D4, #-3.0

          if operandNum == 2:
            contentStr = "%s%s%s" % (srcOperandStr, Instruction.toStr, dstOperandStr)
            # print("contentStr=%s" % contentStr)
          elif operandNum > 2:
            # TODO: add case for operand > 2
            print("TODO: add support operand > 2 of MOV/FMOV")
        elif self.isAdd() or self.isFadd():
          # <Instruction: 0x10235D574: ADD X0, X19, X8; location>
          # # print("is ADD: self=%s" % self)
          # instName = self.name
          # # print("instName=%s" % instName)
          # instOperandList = self.operands
```

```
      # # print("instOperandList=%s" % Operand.listToStr(instOperandList))
      if operandNum == 3:
        # <Instruction: 0x10235D574: ADD X0, X19, X8; location>
        extracOperand = self.operands[2]
        # print("extracOperand=%s" % extracOperand)
        extraOperandStr = extracOperand.contentStr
        # print("extraOperandStr=%s" % extraOperandStr)
        contentStr = "%s%s%s%s%s" % (srcOperandStr, Instruction.addStr, extraOperandStr,
 Instruction.toStr, dstOperandStr)

      # TODO: add case operand == 2
    elif self.isLdr():
      # LDR X0, [SP,#arg_18];
      if operandNum == 2:
        contentStr = "%s%s%s" % (srcOperandStr, Instruction.toStr, dstOperandStr)
      elif operandNum > 2:
        # TODO: add case for operand > 2
        print("TODO: add support operand > 2 of LDR")
    elif self.isStr():
      # STR XZR, [X19,X8]
      if operandNum == 2:
        contentStr = "%s%s%s" % (dstOperandStr, Instruction.toStr, srcOperandStr)
      elif operandNum > 2:
        # TODO: add case for operand > 2
        print("TODO: add support operand > 2 of STR")
    elif self.isStp():
      # <Instruction: 0x10235D6B4: STP X8, X9, [SP,#arg_18]>
      if operandNum == 3:
        srcOperand1 = self.operands[0]
        if isDebug:
          print("srcOperand1=%s" % srcOperand1)
        srcOperand1Str = srcOperand1.contentStr
        if isDebug:
          print("srcOperand1Str=%s" % srcOperand1Str)
        srcOperand2 = self.operands[1]
        if isDebug:
          print("srcOperand2=%s" % srcOperand2)
        srcOperand2Str = srcOperand2.contentStr
        if isDebug:
          print("srcOperand2Str=%s" % srcOperand2Str)

        dstOperand = self.operands[2]
        if isDebug:
          print("dstOperand=%s" % dstOperand)
        dstOperandStr = dstOperand.contentStr
        if isDebug:
          print("dstOperandStr=%s" % dstOperandStr)

        contentStr = "%s%s%s%s" % (srcOperand1Str, srcOperand2Str, Instruction.toStr, d
stOperandStr)
    elif self.isLdp():
      # <Instruction: 0x10235D988: LDP D0, D1, [X8]>
      # <Instruction: 0x10235D98C: LDP D2, D3, [X8,#0x10]>
      if operandNum == 3:
        dstOperand1 = self.operands[0]
        if isDebug:
          print("dstOperand1=%s" % dstOperand1)
        dstOperand1Str = dstOperand1.contentStr
        if isDebug:
          print("dstOperand1Str=%s" % dstOperand1Str)
        dstOperand2 = self.operands[1]
        if isDebug:
          print("dstOperand2=%s" % dstOperand2)
        dstOperand2Str = dstOperand2.contentStr
        if isDebug:
          print("dstOperand2Str=%s" % dstOperand2Str)

        srcOperand = self.operands[2]
        if isDebug:
          print("srcOperand=%s" % srcOperand)
        srcOperandStr = srcOperand.contentStr
        if isDebug:
          print("srcOperandStr=%s" % srcOperandStr)

        contentStr = "%s%s%s%s" % (srcOperandStr, Instruction.toStr, dstOperand1Str, ds
tOperand2Str)

    # TODO: add other Instruction support: SUB/STR/...
    if isDebug:
      print("contentStr=%s" % contentStr)
    return contentStr

  def isMov(self):
    return self.isInst("MOV")

  def isFmov(self):
    return self.isInst("FMOV")

  def isRet(self):
    return self.isInst("RET")

  def isB(self):
    return self.isInst("B")

  def isBr(self):
    return self.isInst("BR")

  def isBranch(self):
    # TODO: support more: BRAA / ...
    return self.isB() or self.isBr()

  def isAdd(self):
    return self.isInst("ADD")

  def isFadd(self):
    return self.isInst("FADD")

  def isSub(self):
    return self.isInst("SUB")

  def isStr(self):
    return self.isInst("STR")
  def isStp(self):
    return self.isInst("STP")

  def isLdp(self):
    return self.isInst("LDP")

  def isLdr(self):
    return self.isInst("LDR")
```

IDA常用插件脚本

crifan的IDA插件脚本

我自己=Crifan的IDA插件脚本有：

- （从别人的脚本）优化后的：从IDA中导出ObjC的block符号表
    - ida_search_block.py
- 自动给IDA中符号重命名=优化符号命名
    - AutoRename
- 从IDA中导出符号表
    - exportIDASymbol.py

其他的IDA插件脚本

常见问题

附录

下面列出相关参考资料。

参考资料

- IDAPython documentation
- IDAPython and Python 3 – Hex Rays
- Turning off IDA 6.x compatibility in IDAPython – Hex Rays
- Porting guide for changes in IDAPython-on-Python-3 APIs – Hex Rays
- Reverse Engineering Tips — IDA Python | by Thomas Roccia | SecurityBreak
- IDAPython documentation
- idapython/src: IDAPython project for Hex-Ray's IDA Pro
- idaapi
-