GETTING STARTED

Basic Usage

Basic Usage

In this document, we'll explore the essentials of IDA capabilities to kickstart your journey and disassemble your first binary file.

Prerequisites

Your IDA instance is installed and running.

Before you begin

What files and processors are supported?

IDA natively recognizes plenty of file formats and processors.

If you later realize that's not enough, you can always use one of our community plugins that add additional formats or processor types or try to write your own with C++ SDK.

What are IDA database files?

IDA stores the analysis results in the IDA Database files (called IDB), with the extension `.i64`. This allows you to save your work and continue from the same point later. After loading a file at the beginning, IDA does not require access to the binary.

Any modifications you make are saved in the database and do not affect the original executable file.

Dive deeper

- **Blog**: 📝 Check what exactly IDB contains in Igor's tip of the week about IDA database.

What decompilers can I work with?

IDA provides decompilers designed to work with multiple processor architectures. The number of decompilers and their type (local or remote) available in your IDA instance depends on your chosen product and subscription plan and affects your ability to produce C-like pseudocode.

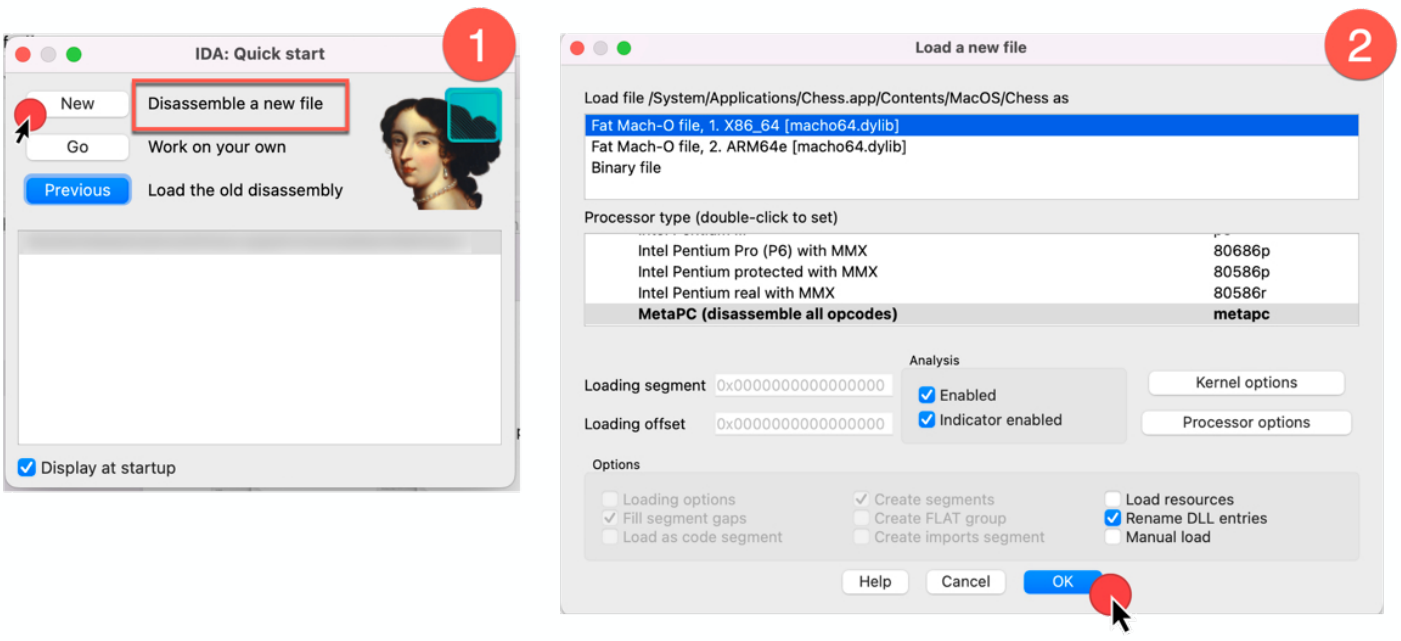
Where can I find exemplary binaries to work with?

Check CrackMe, from where you can download executable files to test your reverse engineering skills.

Part 1: Loading your file

When you launch IDA, you will see a Quick Start dialog that offers three ways to continue. For now, we'll focus on loading a new file and proceeding to disassembly results.

1. Launch IDA and in the **Quick start** dialog (1), click **New**.
2. Specify the path for your binary file.
3. In the **Load a new file** dialog (2), IDA presents loaders that are suited to deal with a selected file. Accepting the loader default selection and then the processor type is a good strategy for beginners. Click **OK** to confirm your selection.



IDA: Quick start

- New — Disassemble a new file
- Go — Work on your own
- Previous — Load the old disassembly

☑ Display at startup

Load a new file

Load file /System/Applications/Chess.app/Contents/MacOS/Chess as

- Fat Mach-O file, 1. X86_64 [macho64.dylib]
- Fat Mach-O file, 2. ARM64e [macho64.dylib]
- Binary file

Processor type (double-click to set)

Intel Pentium Pro (P6) with MMX	80686p
Intel Pentium protected with MMX	80586p
Intel Pentium real with MMX	80586r
MetaPC (disassemble all opcodes)	**metapc**

Loading segment 0x0000000000000000

Loading offset 0x0000000000000000

Analysis

☑ Enabled
☑ Indicator enabled

Kernel options
Processor options

Options

☐ Loading options
✓ Fill segment gaps
☐ Load as code segment
✓ Create segments
☐ Create FLAT group
☐ Create imports segment
☐ Load resources
☑ Rename DLL entries
☐ Manual load

Quick start

4. IDA begins autoanalysis of your binary file.

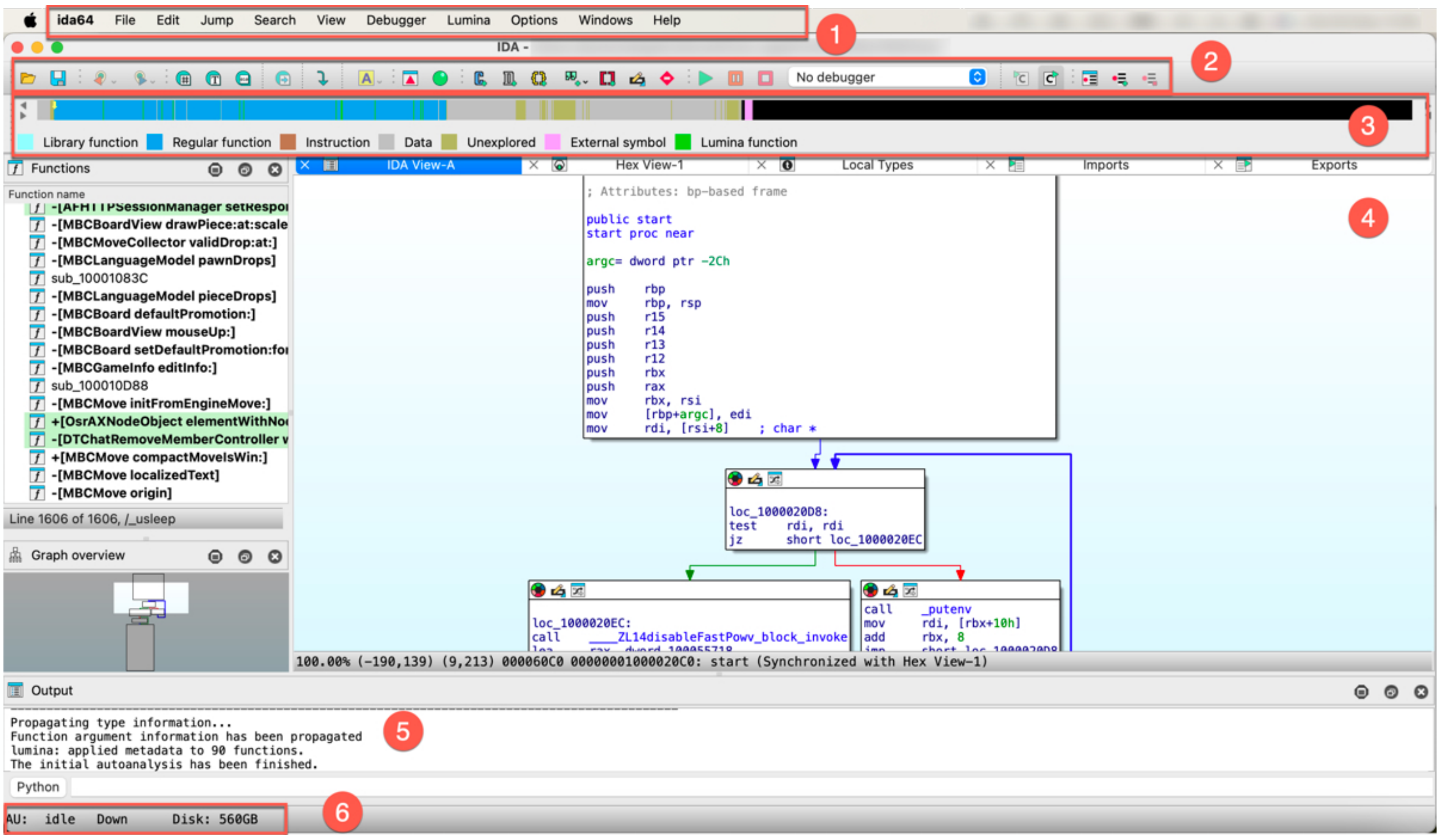
After completion, you will be present with the default IDA desktop layout, that we'll describe in the next part.

Dive deeper

- **Video**: 📹 Watch different ways of loading files in our channel.

Part 2: UI overview

After autoanalysis is done, you'll see the main IDA desktop with the initial results. Let's examine the default desktop layout and commonly used UI elements.



ida64 File Edit Jump Search View Debugger Lumina Options Windows Help
IDA -
No debugger
🟦 Library function 🟦 Regular function 🟫 Instruction ▨ Data 🟨 Unexplored 🟪 External symbol 🟩 Lumina function
Functions
IDA View-A
Hex View-1
Local Types
Imports
Exports
Graph overview
Output
Python
1
2
3
4
5
6

Function name
-[AFHTTPSessionManager setRespo
-[MBCBoardView drawPiece:at:scale
-[MBCMoveCollector validDrop:at:]
-[MBCLanguageModel pawnDrops]
sub_10001083C
-[MBCLanguageModel pieceDrops]
-[MBCBoard defaultPromotion:]
-[MBCBoardView mouseUp:]
-[MBCBoard setDefaultPromotion:fo
-[MBCGameInfo editInfo:]
sub_100010D88
-[MBCMove initFromEngineMove:]
+[OsrAXNodeObject elementWithNo
-[DTChatRemoveMemberController v
+[MBCMove compactMoveIsWin:]
-[MBCMove localizedText]
-[MBCMove origin]

Line 1606 of 1606, /_usleep

```
; Attributes: bp-based frame

public start
start proc near

argc= dword ptr -2Ch

push    rbp
mov     rbp, rsp
push    r15
push    r14
push    r13
push    r12
push    rbx
push    rax
mov     rbx, rsi
mov     [rbp+argc], edi
mov     rdi, [rsi+8]    ; char *

loc_1000020D8:
test    rdi, rdi
jz      short loc_1000020EC

loc_1000020EC:
call    ____ZL14disableFastPowv_block_invoke

call    _putenv
mov     rdi, [rbx+10h]
add     rbx, 8
```

```
100.00% (-190,139) (9,213) 000060C0 00000001000020C0: start (Synchronized with Hex View-1)
```

```
Propagating type information...
Function argument information has been propagated
lumina: applied metadata to 90 functions.
The initial autoanalysis has been finished.
```

```
AU:  idle   Down       Disk: 560GB
```

UI overview

1. Main menu bar (1)
2. Toolbar (2)
3. Navigation band (3)
4. Subviews (4)
5. Output (5)
6. Status bar (6)

Main menu bar

The main menu bar provides quick access to essential features. Moreover, almost all menu commands can be quickly accessible via customizable shortcuts.

Main Menu Bar

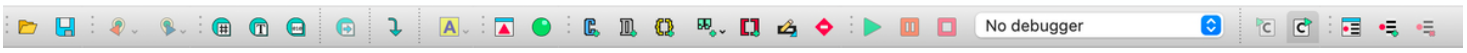
For a handy cheatsheet of all commands and their hotkeys, check **Options → Show command palette...**.

Dive deeper

- **Docs**: 📖 Check our User Guide for a comprehensive description of all menu items.

Toolbar

Below the main menu bar, you will see a toolbar with icons that give you quick access to common functionalities (available also via the main menu/shortcuts). It has just one line by default, but you can customize it by adding or rearranging your actions.



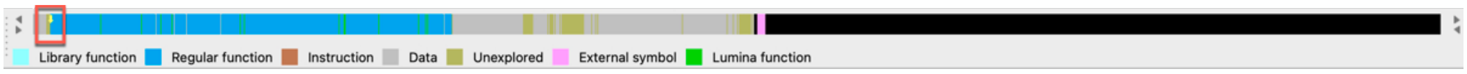
Toolbar

Dive deeper

- **Video**: 📹 Curious about practical ways to set up your toolbar? Watch our video tutorial.

Navigation band

The navigation band shows the graphical representation of the analyzed binary file and gives a short overview of its contents and which areas may need your attention. The yellow arrow (indicator) shows where the cursor is currently positioned in the disassembly view.



Navigation band

As you'll soon recognize, the colors used in the nav band match those in other views.

Dive deeper

- **Blog**: 📝 A detailed navigation band overview with the full colors legend you can found in Igor's tip of the week.

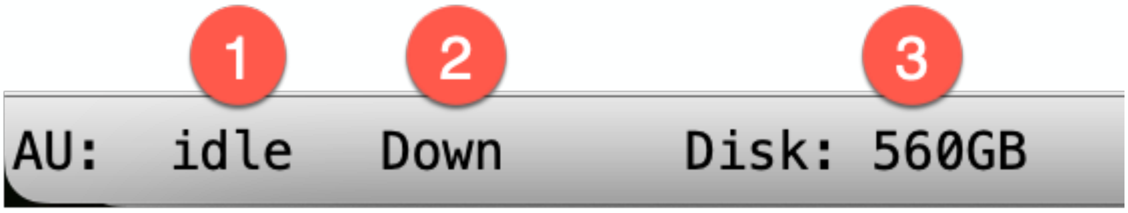
Output

The output window is a place where various messages and logs are displaying, often descrybing what currently IDA is doing, like analyzing data or running a script. In the CLI box you can type commands in IDC language or IDAPython.

Status bar

At the bottom left corner of the IDA window, you can see the status bar, which contains:

- analysis indicator `AU`, which shows the actual status of autoanalysis (1). In our case, it is `idle`, which means the autoanalysis is already finished.
- search direction indicator (2)
- remaining free disk space (3)



Status bar

Right-clicking on the status bar brings up a context menu that allows you to **reanalyze the program**.

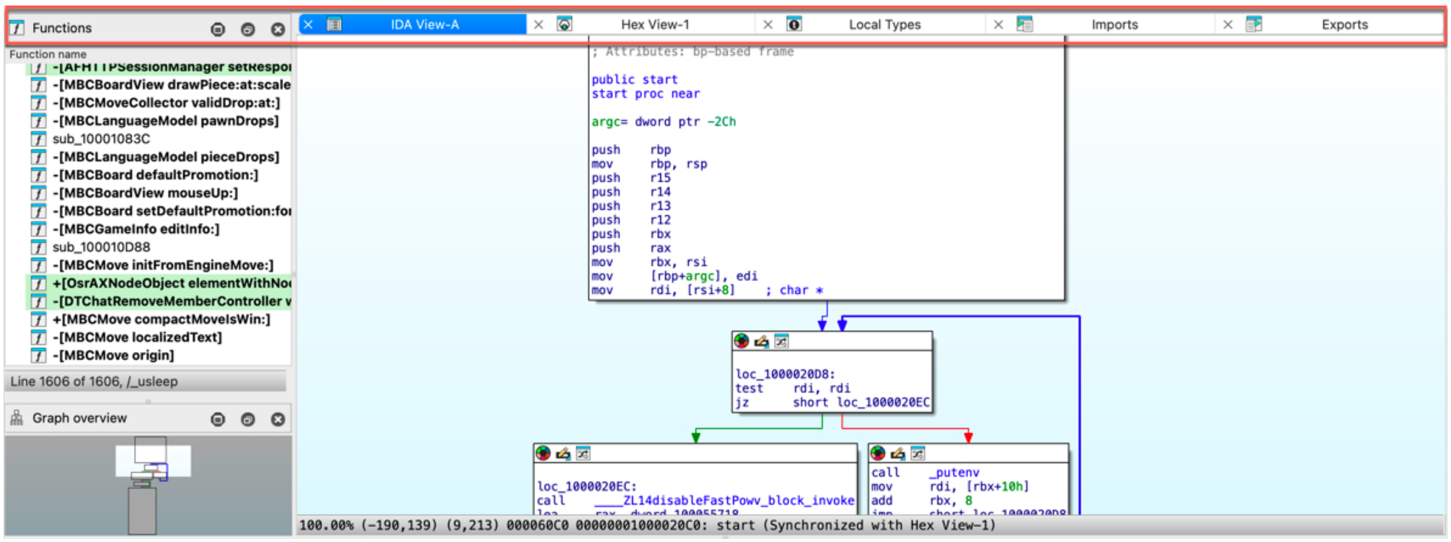
Dive deeper

- **Docs:** 📖 To check all possible values and their meaning, take a look at analysis options.

Subviews

The **subviews** are one of the most prominent parts of your everyday work with IDA. These additional views (behaving like tabs) give a different perspective and information on the binary file, but the number of native IDA subviews may be a bit overwhelming. Here, we will focus on the most versatile and common subviews for beginners, where you'll spend most of the time, like:

- IDA View
- Pseudocode
- Hex Dump View
- Local Types
- Functions View



```
-[AFHTTPSessionManager setRespo
-[MBCBoardView drawPiece:at:scale
-[MBCMoveCollector validDrop:at:]
-[MBCLanguageModel pawnDrops]
sub_10001083C
-[MBCLanguageModel pieceDrops]
-[MBCBoard defaultPromotion:]
-[MBCBoardView mouseUp:]
-[MBCBoard setDefaultPromotion:fo
-[MBCGameInfo editInfo:]
sub_100010D88
-[MBCMove initFromEngineMove:]
+[OsrAXNodeObject elementWithNo
-[DTChatRemoveMemberController v
+[MBCMove compactMoveIsWin:]
-[MBCMove localizedText]
-[MBCMove origin]
Line 1606 of 1606, /_usleep
```

```
; Attributes: bp-based frame

public start
start proc near

argc= dword ptr -2Ch

push    rbp
mov     rbp, rsp
push    r15
push    r14
push    r13
push    r12
push    rbx
push    rax
mov     rbx, rsi
mov     [rbp+argc], edi
mov     rdi, [rsi+8]    ; char *

loc_1000020D8:
test    rdi, rdi
jz      short loc_1000020EC

loc_1000020EC:
call    ___ZL14disableFastPoww_block_invoke

call    _putenv
mov     rdi, [rbx+10h]
add     rbx, 8

100.00% (-190,139) (9,213) 000060C0 00000001000020C0: start (Synchronized with Hex View-1)
```

Subviews

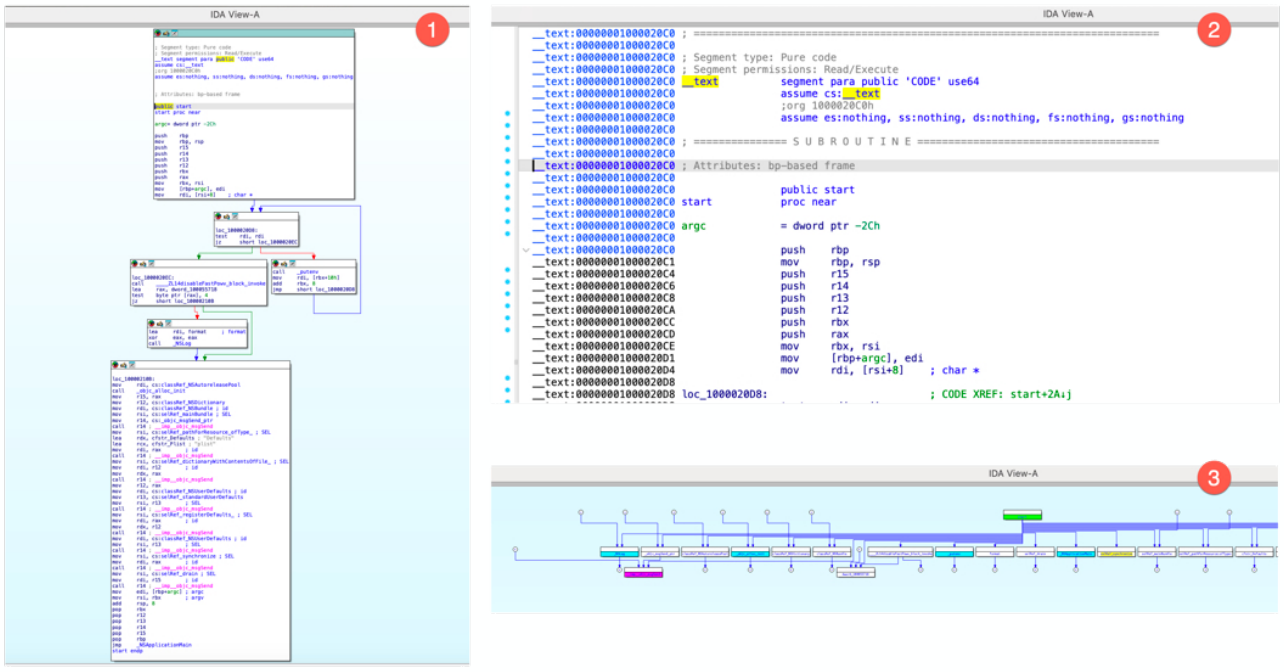
IDA View / Disassembly Window

When autoanalysis is done, you will see a **graph view** inside an IDA View by default. This flowchart graph should help you to understand the flow of the functions.

The graph view is available only for the part of the binary that IDA has recognized as functions.

IDA view has three modes:

- **graph view** (1), that shows instructions grouped in blocks,
- **linear view** (2), that lists all instructions and data in order of their addresses,
- and **proximity view** (3), which allows you to see relations between functions, global variables, and other parts of the program.



```
__text:00000001000020C0 ; ===========================================================================
__text:00000001000020C0
__text:00000001000020C0 ; Segment type: Pure code
__text:00000001000020C0 ; Segment permissions: Read/Execute
__text:00000001000020C0 __text          segment para public 'CODE' use64
__text:00000001000020C0                 assume cs:__text
__text:00000001000020C0                 ;org 1000020C0h
__text:00000001000020C0                 assume es:nothing, ss:nothing, ds:nothing, fs:nothing, gs:nothing
__text:00000001000020C0
__text:00000001000020C0 ; =============== S U B R O U T I N E =======================================
__text:00000001000020C0
__text:00000001000020C0 ; Attributes: bp-based frame
__text:00000001000020C0
__text:00000001000020C0                 public start
__text:00000001000020C0 start           proc near
__text:00000001000020C0
__text:00000001000020C0 argc            = dword ptr -2Ch
__text:00000001000020C0
__text:00000001000020C0                 push    rbp
__text:00000001000020C1                 mov     rbp, rsp
__text:00000001000020C4                 push    r15
__text:00000001000020C6                 push    r14
__text:00000001000020C8                 push    r13
__text:00000001000020CA                 push    r12
__text:00000001000020CC                 push    rbx
__text:00000001000020CD                 push    rax
__text:00000001000020CE                 mov     rbx, rsi
__text:00000001000020D1                 mov     [rbp+argc], edi
__text:00000001000020D4                 mov     rdi, [rsi+8]    ; char *
__text:00000001000020D8
__text:00000001000020D8 loc_1000020D8:                          ; CODE XREF: start+2A↓j
```

IDA view modes

Press `Space` to switch between graph and linear mode. Provimity view is available from the context menu in IDA view.

Dive deeper

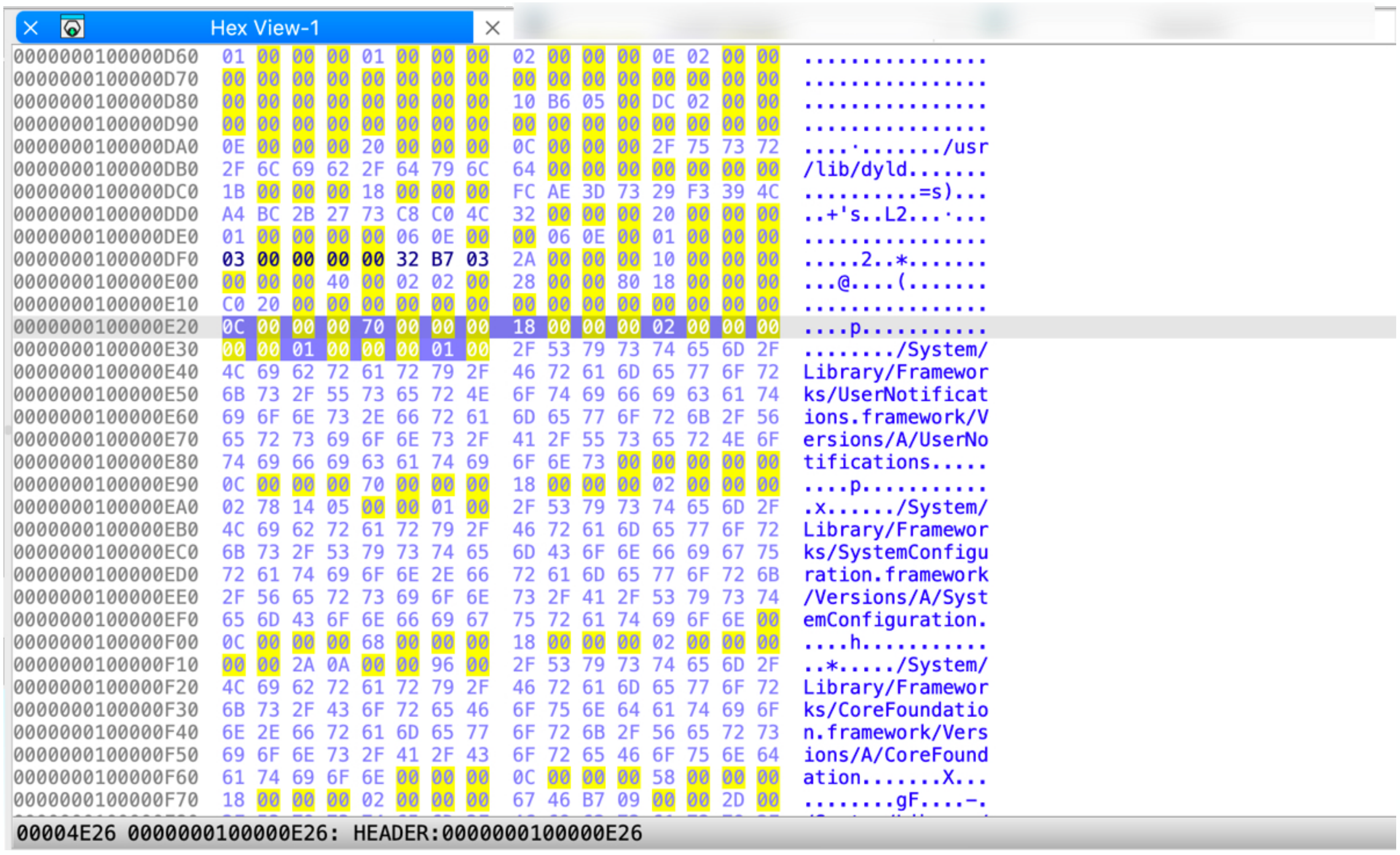
- **Video**: 📹 Check our video tutorial covering the basics of graph view.
- **Blog**: 📝 Read the graph mode overview in Igor's tip of the week.

Hex View Window

In hex view, you can see the raw bytes of the program's instructions.

There are two ways of highlighting the data in this view:

1. **Text match highlight**, which shows matches of the selected text anywhere in the views.
2. **Current item highlight**, which shows the bytes group constituting the current item.


Hex View-1

```
0000000100000D60  01 00 00 00 01 00 00 00  02 00 00 00 0E 02 00 00  ................
0000000100000D70  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0000000100000D80  00 00 00 00 00 00 00 00  10 B6 05 00 DC 02 00 00  ................
0000000100000D90  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0000000100000DA0  0E 00 00 00 20 00 00 00  0C 00 00 00 2F 75 73 72  ....·......./usr
0000000100000DB0  2F 6C 69 62 2F 64 79 6C  64 00 00 00 00 00 00 00  /lib/dyld.......
0000000100000DC0  1B 00 00 00 18 00 00 00  FC AE 3D 73 29 F3 39 4C  ..........=s)...
0000000100000DD0  A4 BC 2B 27 73 C8 C0 4C  32 00 00 00 20 00 00 00  ..+'s..L2...·...
0000000100000DE0  01 00 00 00 00 06 0E 00  00 06 0E 00 01 00 00 00  ................
0000000100000DF0  03 00 00 00 00 32 B7 03  2A 00 00 00 10 00 00 00  .....2..*.......
0000000100000E00  00 00 00 40 00 02 02 00  28 00 00 80 18 00 00 00  ...@....(.......
0000000100000E10  C0 20 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0000000100000E20  0C 00 00 00 70 00 00 00  18 00 00 00 02 00 00 00  ....p...........
0000000100000E30  00 00 01 00 00 00 01 00  2F 53 79 73 74 65 6D 2F  ......../System/
0000000100000E40  4C 69 62 72 61 72 79 2F  46 72 61 6D 65 77 6F 72  Library/Framewor
0000000100000E50  6B 73 2F 55 73 65 72 4E  6F 74 69 66 69 63 61 74  ks/UserNotificat
0000000100000E60  69 6F 6E 73 2E 66 72 61  6D 65 77 6F 72 6B 2F 56  ions.framework/V
0000000100000E70  65 72 73 69 6F 6E 73 2F  41 2F 55 73 65 72 4E 6F  ersions/A/UserNo
0000000100000E80  74 69 66 69 63 61 74 69  6F 6E 73 00 00 00 00 00  tifications.....
0000000100000E90  0C 00 00 00 70 00 00 00  18 00 00 00 02 00 00 00  ....p...........
0000000100000EA0  02 78 14 05 00 00 01 00  2F 53 79 73 74 65 6D 2F  .x....../System/
0000000100000EB0  4C 69 62 72 61 72 79 2F  46 72 61 6D 65 77 6F 72  Library/Framewor
0000000100000EC0  6B 73 2F 53 79 73 74 65  6D 43 6F 6E 66 69 67 75  ks/SystemConfigu
0000000100000ED0  72 61 74 69 6F 6E 2E 66  72 61 6D 65 77 6F 72 6B  ration.framework
0000000100000EE0  2F 56 65 72 73 69 6F 6E  73 2F 41 2F 53 79 73 74  /Versions/A/Syst
0000000100000EF0  65 6D 43 6F 6E 66 69 67  75 72 61 74 69 6F 6E 00  emConfiguration.
0000000100000F00  0C 00 00 00 68 00 00 00  18 00 00 00 02 00 00 00  ....h...........
0000000100000F10  00 00 2A 0A 00 00 96 00  2F 53 79 73 74 65 6D 2F  ..*...../System/
0000000100000F20  4C 69 62 72 61 72 79 2F  46 72 61 6D 65 77 6F 72  Library/Framewor
0000000100000F30  6B 73 2F 43 6F 72 65 46  6F 75 6E 64 61 74 69 6F  ks/CoreFoundatio
0000000100000F40  6E 2E 66 72 61 6D 65 77  6F 72 6B 2F 56 65 72 73  n.framework/Vers
0000000100000F50  69 6F 6E 73 2F 41 2F 43  6F 72 65 46 6F 75 6E 64  ions/A/CoreFound
0000000100000F60  61 74 69 6F 6E 00 00 00  0C 00 00 00 58 00 00 00  ation.......X...
0000000100000F70  18 00 00 00 02 00 00 00  67 46 B7 09 00 00 2D 00  ........gF....-.
00004E26 0000000100000E26: HEADER:0000000100000E26
```

Hex view

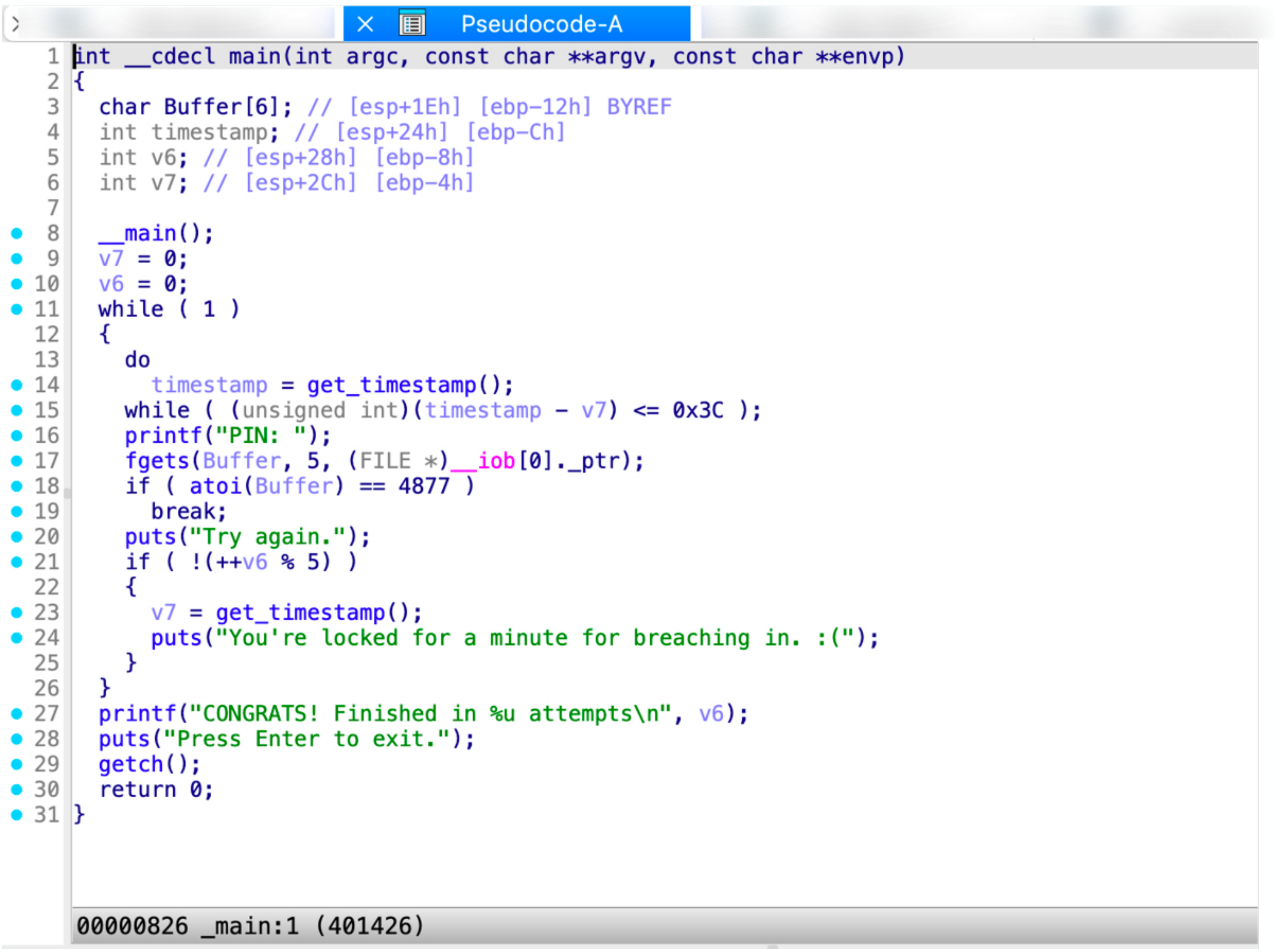
The IDA view, pseudocode, and hex view can be synchronized, meaning that they highlight the same part of the analyzed program, and changes made inside one of the views are visible in the others.

Dive deeper

- **Video**: 📹 Listen about hex view and others in our video tutorial.
- **Blog**: 📝 Detailed overview of the hex view you can read in Igor's tip of the week.

Pseudocode Window

Generated by the famous `F5` shortcut, the pseudocode shows the assembly language translated into human-readable, C-like pseudocode. Click `Tab` to jump right into the Pseudocode view.



```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char Buffer[6]; // [esp+1Eh] [ebp-12h] BYREF
  int timestamp; // [esp+24h] [ebp-Ch]
  int v6; // [esp+28h] [ebp-8h]
  int v7; // [esp+2Ch] [ebp-4h]

  __main();
  v7 = 0;
  v6 = 0;
  while ( 1 )
  {
    do
      timestamp = get_timestamp();
    while ( (unsigned int)(timestamp - v7) <= 0x3C );
    printf("PIN: ");
    fgets(Buffer, 5, (FILE *)__iob[0]._ptr);
    if ( atoi(Buffer) == 4877 )
      break;
    puts("Try again.");
    if ( !(++v6 % 5) )
    {
      v7 = get_timestamp();
      puts("You're locked for a minute for breaching in. :(");
    }
  }
  printf("CONGRATS! Finished in %u attempts\n", v6);
  puts("Press Enter to exit.");
  getch();
  return 0;
}
```

```
00000826 _main:1 (401426)
```

Pseudocode Window

Local Types Window

This view shows the high-level types used in databases, like structs or enums.

Dive deeper

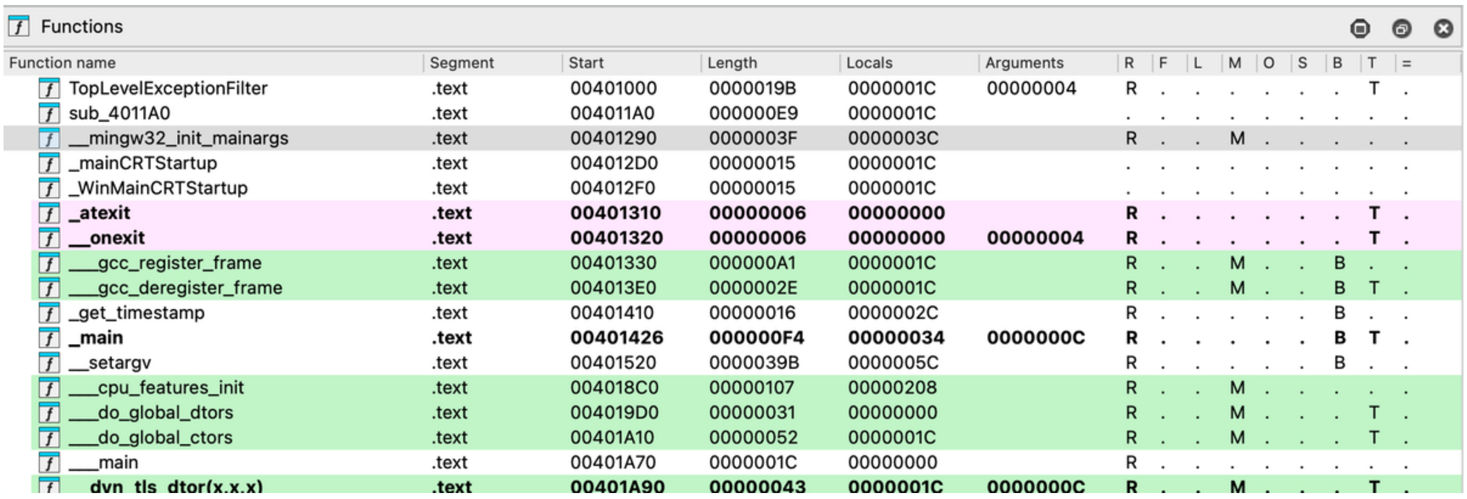
- **Docs**: 📖 Check our manual giving an overview of Local Types window.

Functions Window

This window displays all the functions recognized by IDA, along with key details for each:

- **Function name**
- **Segment** the segment that contains the function
- **Start**: the function starting address
- **Length**: the size of the function in bytes
- **Local**: the amount of stack space taken by local variables
- **Arguments**: the amount of stack space taken by arguments

By default, the entire window is not visible, so you may scroll horizontally to see the hidden elements. As you proably noticed, the colors in Functions window match the colors in navigation band; in our example, green higlightning shows functions recognized by Lumina.



Function name	Segment	Start	Length	Locals	Arguments	R	F	L	M	O	S	B	T	=
TopLevelExceptionFilter	.text	00401000	0000019B	0000001C	00000004	R	.	.	.	.	.	.	T	.
sub_4011A0	.text	004011A0	000000E9	0000001C		.	.	.	.	.	.	.	.	.
__mingw32_init_mainargs	.text	00401290	0000003F	0000003C		R	.	.	M	.	.	.	.	.
_mainCRTStartup	.text	004012D0	00000015	0000001C		.	.	.	.	.	.	.	.	.
_WinMainCRTStartup	.text	004012F0	00000015	0000001C		.	.	.	.	.	.	.	.	.
_atexit	**.text**	**00401310**	**00000006**	**00000000**		**R**	.	.	.	.	.	.	**T**	.
__onexit	**.text**	**00401320**	**00000006**	**00000000**	**00000004**	**R**	.	.	.	.	.	.	**T**	.
___gcc_register_frame	.text	00401330	000000A1	0000001C		R	.	.	M	.	.	B	.	.
___gcc_deregister_frame	.text	004013E0	0000002E	0000001C		R	.	.	M	.	.	B	T	.
_get_timestamp	.text	00401410	00000016	0000002C		R	.	.	.	.	.	B	.	.
_main	**.text**	**00401426**	**000000F4**	**00000034**	**0000000C**	**R**	.	.	.	.	.	**B**	**T**	.
__setargv	.text	00401520	0000039B	0000005C		R	.	.	.	.	.	B	.	.
___cpu_features_init	.text	004018C0	00000107	00000208		R	.	.	M	.	.	.	.	.
___do_global_dtors	.text	004019D0	00000031	00000000		R	.	.	M	.	.	.	T	.
___do_global_ctors	.text	00401A10	00000052	0000001C		R	.	.	M	.	.	.	T	.
___main	.text	00401A70	0000001C	00000000		R	.	.	.	.	.	.	.	.
__dyn_tls_dtor(x,x,x)	**.text**	**00401A90**	**00000043**	**0000001C**	**0000000C**	**R**	.	.	**M**	.	.	.	**T**	.

Functions view

This view is read-only, but you can automatically synchronize the function list with the IDA view, pseudocode, or hex view. Click to open the context menu and select **Turn on synchronization**.

Dive deeper

- **Docs**: 📖 Read the manual explaining all of the function window columns in detail.
- **Video**: 📹 Watch our video tutorial exploring the funcions view.

Part 3: Basic navigation

A crucial step in mastering IDA is learning how to navigate quickly to specific locations in the output. To help you get started, we'll cover essential commands and hotkeys commonly used for efficient navigation in IDA.

Double-click and jump to the location

When you double-click on an item, such as a name or address, IDA automatically jumps to that location and relocate the display.

Jump to address

1. Go to **Jump → Jump to address..** or press `G` hotkey
2. Enter the item name or hex address in the dialog box, then click **OK**.

To jump back to the previous position, press `Esc`. To jump to the next position, press `Ctrl + Enter`. You can also navigate using the arrows in the toolbar.

See the list of cross-references

1. Position the cursor on a function or instruction, then go to **Jump → Jump to xref to operand...** or press `X` to see the dialog with listed all cross-references to this identifier.
2. Select an item from the list and click **OK** to jump to that location.

Dive deeper

- **Video**: 📹 Explore the rest of the jump commands in our video tutorial

Part 4: Manipulate your disassembly results

Now that the initial autoanalysis is done and you've mastered the basics of navigation, it's time to explore the basic interactive operations that reveal the true power of IDA in transforming your analysis.

Rename a stack variable

One of the first steps you might take is to enhance readability by assigning meaningful names to local or global variables, but also functions, registers and other objects that IDA initially assigned a dummy name.

1. In the IDA View, right-click on the variable you want to rename and click **Rename** or press `N` when the variable is cursor-highlighted.
2. In the newly opened dialog, insert a new name and click **OK**.

If at any point you want to go back to the original dummy name given by IDA, leave the field blank and click **OK**. It will reset the name to the default one.

ⓘ Once you change the name, IDA will propagate the changes through the decompiler and Pseudocode view.

Dive deeper

- **Docs**: 📖 Check the details on renaming items in the User Guide
- **Video**: 📹 Watch our step-by-step tutorial on renaming techniques.
- **Blog**: 📝 Check Igor's tips of the week for expert advice on renaming.

Add a comment

Adding comments may be a useful way to annotate your work.

1. Highlight the line where you want to insert a comment and press `:`.
2. In the dialog box, type your comment (you can use multiple lines) and click **OK**. This will add a regular (non-repeatable) comment to the location.

ⓘ If you want to add a repeatable comment in every location that refers to the original comment, press ';'.

Dive deeper

- **Video**: 📹 Watch our tutorial about commenting.

Part 5: Customizing IDA

Nearly every UI element is customizable, allowing you to rearrange and align widgets to suit your habits. You can save your personalized desktop layout by going to **Windows → Save desktop**.

Most of the basic appearance you can change under **Options** menu.

- To change the colors or theme, go to **Options → Colors**.
- To change the font, go to **Options → Fonts**.

If you need more control over customization settings, you may check the IDA configuration files.

Part 6: Debug your file

If you are ready to delve into dynamic analysis and start debugging your programs, here are some key steps to get you started:

1. **Select the right debugger and complete the setup**: Go to **Debugger → Select debugger...** and pick up one of the avaliable debuggers. Under **Debugger → Debugger options**, you can configure the setup in detail.
2. **Add breakpoints**: Right-click on the line where you want to stop the execution and select **Add breakpoint** from the context menu, or press `F2`.
3. **Start the process**: Run the debuggin session by pressing `F9` or click a green arrow on the tooltip.

Dive Deeper

- **Docs**: 📖 Read our User Guide for local and remote debugging manuals, or check step-by-step tutorials for specific debuggers.

Part 7: Install a plugin

One of the most common way of extanding IDA capabilities is to use on of our community-developed plugins.

Where can I find IDA plugins?

You can find a variety of plugins in the official Hex-Rays plugin repository

Installing your plugin

For this guide purposes, we'll walk you through general installation steps.

ⓘ The installation process can vary depending on the plugin and some of them may required installing dependencies or further configuration. Don't hesitate to refer to the specific instructions provided by the plugin author.

Load your plugin

1. Copy your plugin folder to the plugins directory inside your IDA installation directory.
2. Alternatively, you can load the plugin from the command line in IDA by using **File → Script file...** and selecting `app.entry.py` file.

Run your plugin

1. Navigate to **Edit → Plugins → your_plugin_name** or use the assigned hotkey.

You may need to restart IDA to see your plugin in the list.

Dive deeper

- **Docs**: 📖 Want to learn about writing your own plugins? Check our Developer Guide on how to create a plugin in IDAPython or with C++ SDK.

Key hotkeys cheatsheet

Here's a handy list of all of the shortcuts we used so far.

- `Space` Switches between graph and linear mode in the IDA View
- `F5` Generates pseudocode
- `Tab` Jumps into pseudocode View
- `G` Opens **Jump to address** dialog
- `Esc` Jumps back to the previous position
- `Ctrl + Enter` Jumps to the next position
- `X` Shows the list of all cross-references
- `N` Opens dialog to rename the current item
- `;` Adds repeatable comment
- `:` Adds regular comment

Last updated 5 months ago

Was this helpful?