ImaginaryInfinity Calculator

Release 2.13.0

Finian Wright & Connor Sample

Aug 31, 2023

CONTENTS

Contents:

CHAPTER

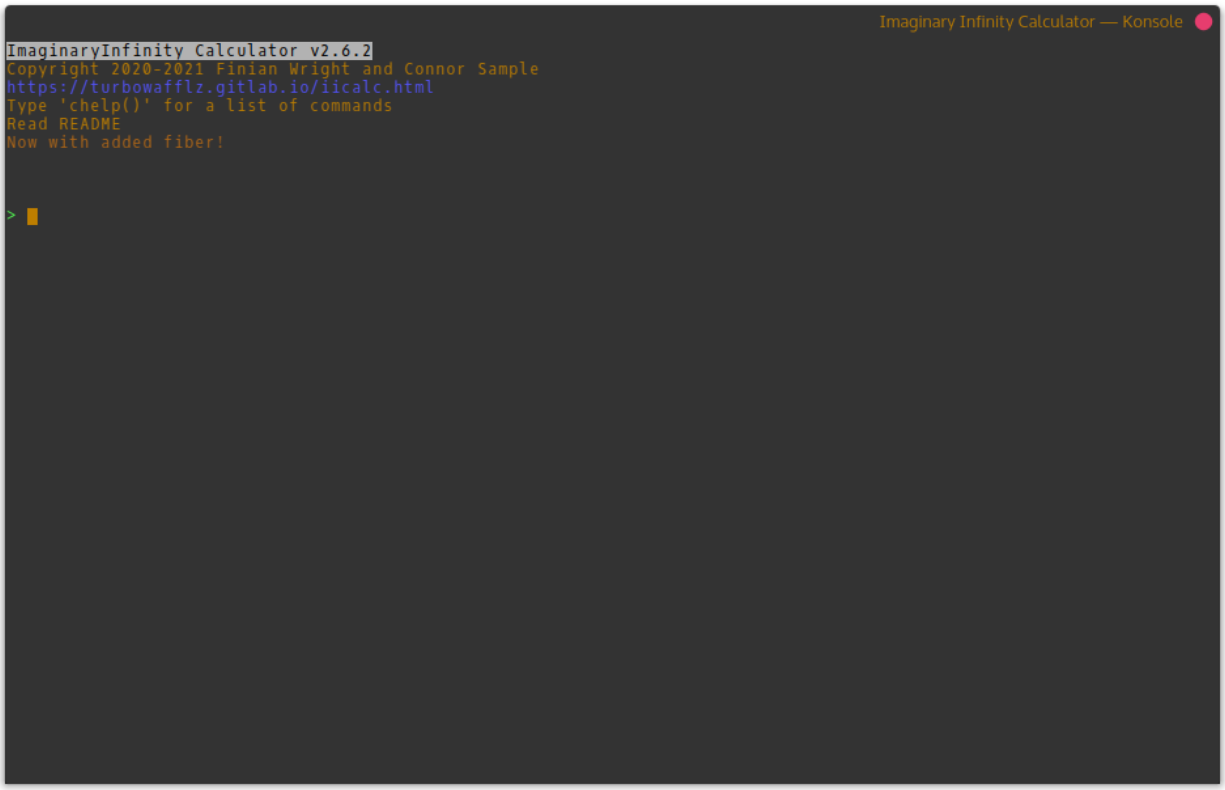
ONE

IMAGINARYINFINITY CALCULATOR

1.1 NOTICE: ImaginaryInfinity Calculator has reached end of life and will no longer receive any updates. We are now working on a new, actually functional calculator, which you can find here: https://gitlab.com/ImaginaryInfinity/squiid-calculator/squiid

ImaginaryInfinity Calculator is a lightweight, but expandable calculator. Its command line interface is designed to resemble that of some graphing calculators. To perform another operation on a result, simply type the operation you wish to perform (eg. "/5" to divide the previous result by 5). To use the previous result in a new calculation, just specify the operation after the new calculation (eg. "5-" to subtract the previous result from 5), or by using the `ans` or `_` variables, like `_ + 3`. Note: commits on the master branch are far between, since we try to do all development on the development branch. If you're interested in testing new features, check out the development branch!

The documentation can also be read at readthedocs.



More screenshots

1.2 Packages

We provide a packaged version of ImaginaryInfinity Calculator for some Linux distributions. They can be found below.

1.2.1 Debian/Ubuntu

iiCalc can be installed on Ubuntu based systems by adding the PPA and installing it via apt, or on Debian/Ubuntu based systems by manually downloading and installing the deb.

PPA:

```
sudo add-apt-repository ppa:imaginaryinfinity/iicalc
sudo apt update

sudo apt install iicalc # master branch
sudo apt install iicalc-beta # development branch
```

Manual installation:

Download (iicalc.deb)

This package can be installed with apt, gdebi, qapt, etc.

1.2.2 Arch Linux

You can install the AUR package or download the package below. To install from the AUR:

```
git clone https://aur.archlinux.org/iicalc.git && cd iicalc
makepkg -si
```

Or you can install it with the below package:

Download (iicalc-any.pkg.tar.zst)

This package can be installed using pacman, with `sudo pacman -U iicalc-any.pkg.tar.zst`

1.2.3 Red Hat/openSUSE

Download (iicalc.rpm)

This package can be installed or updated on Red Hat based systems, like Fedora and CentOS, using rpm, with `sudo dnf install iicalc.rpm` and on openSUSE systems with `sudo zypper install iicalc.rpm`. You can also install it by opening it with a graphical package manager like YaST Software, Discover, GNOME Software, etc.

1.2.4 Windows Installer

Download (iicalc.exe)

Windows may show a window saying "Windows protected your PC." To install the package anyway, click "More Info" and then "Install anyway"

1.2.5 Snap

The iicalc snap can be installed via `snap install iicalc`. To install the development branch/beta channel version, run `snap install --beta iicalc`

1.2.6 AppImage

Download (ImaginaryInfinity_Calculator-x86_64.AppImage)

Just make executable and run on many Linux distributions

1.3 Installation

The advantages of using an installed version of the calculator are that you can launch it from anywhere in a terminal, just by running the `iicalc` command. Installing it also adds a shortcut to your application menu, and if you're on Windows, you have the option to add a desktop shortcut.

1.3.1 Portable Mode

The calculator can be run without any installation on any device that supports Python 3. Just clone the repository and in the directory that you cloned it to, run `python3 main.py`, or replace `python3` with your operating system's python command. (Windows is `py`, Android is `python`, etc.)

1.3.2 Linux/MacOS/Android installation

ImaginaryInfinity Calculator can be installed on Linux by installing the package for your operating system. If your operating system doesn't have a package or you don't want to install a package, you can use the installer script. Just go to the directory where you downloaded the calculator and run `bash installer.sh` in your terminal to install the calculator. The calculator can be easily uninstalled by running `bash uninstaller.sh` in the same directory.

1.3.3 Windows Installation

The calculator can be installed on Windows by downloading and running the windows exe installer or by using the provided powershell script. To install via the powershell script, first search `powershell` in the windows start menu, right click it, and run it as administrator. Then run `Set-ExecutionPolicy Unrestricted -Scope CurrentUser`, and type "Y" to say yes. This makes it so that the installer script can run. Now, navigate to the directory where you downloaded the calculator, and run `.\installer.ps1`. The calculator can be easily uninstalled by opening a powershell session and running `.\uninstaller.ps1` in the same directory.

1.4 Command usage

```
usage: iicalc [-h] [--config CONFIG] [--ensurereqs] [--version]

optional arguments:
  -h, --help            show this help message and exit
  --config CONFIG, -c CONFIG
                        Specify a file path to use as the config file
  --ensurereqs, -e      Ensure that all requirements are satisfied
  --version, -V         Print version and exit
```

1.5 Supported platforms

ImaginaryInfinity Calculator fully supports the following platforms:

- **Linux**
  - Primary development and testing OS. Should have the best support
- **Android**
  - Termux support covers all or most features including installation.

ImaginaryInfinity Calculator has partial support for the following platforms:

- **Haiku**
  - **Command history and line navigation are not currently available in Haiku due to the removal of the readline module**
  - Haiku has fairly good support now and is supported by the installer. There are however a few bugs such as syntax highlighting in documentation not working right
- **MacOS**
  - MacOS 10.15.7 has been tested to work
- **Various BSD OSs**
  - Extensive testing has not been performed, but the calculator should be mostly working on OpenBSD, NetBSD, and FreeBSD
- **Windows**
  - Windows support receives much less testing than other platforms

- **Any other OS that can run Python 3**
  - Start an issue on GitHub and we may improve support for your OS

ImaginaryInfinity Calculator has been tested to work on these platforms:

- Debian/Ubuntu
- Arch Linux
- OpenBSD
- FreeBSD
- NetBSD
- Fedora
- Alpine Linux
- Android (Termux)
- Void Linux
- MacOS Catalina 10.15.7
- Windows (With some issues in the installer; may be fixed in the future)
  - Windows 11
  - Windows 10
  - Windows 8
  - Windows 7 (Python 3.6.9)
- Haiku R1 Beta 2

Python version:

ImaginaryInfinity Calculator has been tested to work on Python 3.6 and above, but support in the future is not guaranteed.

1.6 Plugins

New functionality can easily be added by placing Python files with additional functions in the plugins directory or by downloading plugins from the store. To access a function added by a plugin, type `[plugin].[function]()`. For example, if you wanted to run the function `egg` from the plugin `food`, you would type `food.egg()`. Arguments placed in the parentheses will be passed to the function.

1.6.1 Plugin Documentation

- *Making a standardized plugin or theme*
- *Adding custom config settings*
- *Plugin Guidelines*

Note: Functions in the `core` plugin can be accessed without specifying `core`. ex. `factor(7)` instead of `core.factor(7)`

1.7 Themes

The colors used by the calculator can be modified by themes. Themes are ini files that define the colors the calculator will use and are stored in the `themes` folder. To change the theme used by the calculator, run `settings.configMod("appearance", "theme", "<theme name>")`, or select a theme in the settings editor. One theme is included by default, `dark` for use on terminals with a dark background. If you use a terminal with a light background, you can download our official `light` theme from the store.

1.8 Built in commands:

The following commands are built in to the calculator or added by the "core" plugin:

- `settings.configMod("<section>", "<key>", "<value>")` - Changes a value in the config file.
- `settings.editor()` - Settings editor, not supported on all platforms
- `factor(<number>)` - Shows factor pairs for a number
- `factorList(<number>)` - Returns a list of the factors of a number
- `fancyFactor(<number>)` - Shows factor pairs and their sums and differences for a number
- `iprt('<module>')` - Installs and imports a python module from PyPi
- `isPrime(<number>)` - Checks whether or not a number is is prime
- `isPerfect(<number>)` - Checks whether or not a number's factors add up to twice the starting number
- `restart()` - Restarts iiCalc
- `clear()` or `clear` - Clears the screen
- `sh('<command>')` - Runs a command directly on your computer
- `update()` - Updates the calculator
- `quit()` - Quit ImaginaryInfinity Calculator

1.9 Plugin store

1.9.1 CLI Store

- `pm.update()` - Update the package list, this must be run before plugins can be installed or to check for updates
- `pm.install(*args)` - Installs the specified plugins from the plugin index. (Accepts lists) Example: `pm.install('algebra', 'ptable')`
- `pm.show("<available/installed>")` - List plugins
- `pm.search("<term>")` - Search the plugin index
- `pm.info("<plugin>")` - Show info about a plugin
- `pm.upgrade()` - Install all available updates
- `pm.remove(*args)` - Removes the specified installed plugins. (Accepts lists) Example: `pm.remove('algebra', 'ptable')`
- `pm.installFromFile("<filename>")` - Install a plugin from a local *.icpk file

1.9.2 GUI Store

- `pm.store()` - Runs the GUI version of the plugin store (no longer functional) (`store.store()` is now deprecated and will be removed in a future version)

Note: In the GUI search box, you can specify `type:<type>` anywhere in the query to search for types of plugins. You can add a subquery by specifying it after the type. Example: `type:plugins discord` to search for only plugins with the keyword of discord or `light type:theme` to search for only themes with the keyword of light. Types of plugins include:

- plugin(s)
- theme(s)

1.9.3 Submitting a plugin

The plugin store has been discontinued. You can no longer submit plugins

The following commands accept a second argument to prevent the result from being printed. This is useful when they are used in another function so they don't all get shown to the user:

- `factorList(<number>, [printResult])`
- `isPrime(<number>, [printResult])`
- `isPerfect(<number>, [printResult])`
- `toStd("<value>", [roundVal], [printResult]) - Convert e notation number to standard notation`

printResult can be set to `True` or `False`, and defaults to `True` if not specified

1.10 In-app documentation

Documentation for the calculator and supported plugins can be viewed by running `doc.view("<article>")`. You can also list all articles with `doc.list()`.

CHAPTER

TWO

MAKING AND SUBMITTING A STANDARDIZED PLUGIN OR THEME

This is a guide on how to create and submit a standardized plugin or theme.

2.1 Table of Contents

2.1.1 Making a plugin

Choosing a filename

- First, you make your new plugin file. For this example, we'll be using the `discordrpc` plugin. Make a new python file preferably named to something relevant to your plugin. We'll be choosing the filename `discordrpc.py` for this.

Importing features from core

- If you want to make use of some features from the `core.py` file, you can just import them. For instance, we want to import the current theme if the plugin is printing stuff and the config if the plugin is *configurable*, so at the top of our python file, we will put

```
from systemPlugins.core import config, theme
```

If you want to import anything else from core, you can just add it to the list of things being imported, for instance, if you wanted to import the `restart()` function, it would look like

```
from systemPlugins.core import config, theme, restart
```

Running code on calculator start

If you want to run some code when the calculator is started, for instance, asking the user what their name is to adjust the config, you can just put the code underneath all of your imports in the python file. Example:

```
from systemPlugins.core import config, theme, restart

# Get user's name
name = input("Input your name here: ")
```

Giving answers

When giving answers, it's better to return an answer instead of printing it. This will allow for the resulting calculation to be used in other calculations.

Printing Information

If you are doing anything involving printing, like taking input or just using `print()`, you should make it display with the user's theme's color for that type of output. To see the current color palette, you can type `dev.showPalette()` in the calculator. The available styles can be found below:

- Normal - No styling
- Error - Error Styling
- Warning - Warning Styling
- Important - Important Text Styling
- StartupMessage - Startup Message styling
- Prompt - The color of the prompt (Default prompt is >)

- Link - Hyperlink Style
- Answer - The color that is used when a function returns a value
- Input - The color used for taking input (`input()`)
- Output - General output color

To print something with a specific color, make sure you *import theme from core*, and then you can print like this:

```
#General output
print(theme["styles"]["output"] + "This is some pretty cool general output")

#Taking input
input(theme["styles"]["input"] + "How are you liking this cool theme stuff? ")

#Using a variable
importantVariable = "YOU BETTER PAY ATTENTION TO ME BECAUSE THIS IS VERY IMPORTANT"
print(theme["styles"]["important"] + importantVariable)
```

Handling special events

The calculator will send a signal to plugins when certain events happen. This can be taken advantage of by making a function in your plugin with the same name as the signal name. Some signals take arguments, so that you can do something with the data. Available signal names can be found below:

- `onRestart` - Activated when the calculator restarts
- `onPluginsLoaded` - Activated when all plugins have been loaded
- `onLinuxStart` - Activated when the calculator is started on a Linux operating system
- `onHaikuStart` - Activated when the calculator is started on Haiku
- `onWindowsStart` - Activated when the calculator is started on Windows
- `onMacStart` - Activated when the calculator is started on MacOS
- `onUnknownStart` - Activated when the calculator is started on an unknown operating system
- `onStarted` - Activated once calculator is fully started
- `onReady` - Activated once calculator is ready for user input
- `onInput` - Activated when user inputs something, like a calculation (passes the input as an argument)
- `onError` - Activated on error (passes the exception type as an argument)
- `onPrintAnswer` - Activated when the answer is printed
- `onEofExit` - Activated when a user exits the calculator by pressing Ctrl + D
- `onFatalError` - Activated when the calculator encountered a fatal error and cannot continue (passes the exception type as an argument)
- `onSettingsSaved` - Activated when the settings have been saved and the config file has been updated

An example of the use of these signals can be found here:

```
#This is in your plugin file
def onLinuxStart():
    print(theme["styles"]["output"] + "Hey, you\'re pretty cool, you use linux.")
```

(continues on next page)

(continued from previous page)

```
def onWindowsStart():
   print(theme["styles"]["output"] + "Ew, Windows")

#Since this passes an argument, you must add an argument in the function
def onError(err):
   if err == NameError:
            print(theme["styles"]["error"] + "Whatever you tried to do includes an␣
↪undefined variable")
```

Debugging your plugin

While debugging your plugin a simple error type may not be enough; you may need a traceback. To get the full traceback, you can change `debug` in the `dev` section of the config file to `true`. You can do this manually, with `settings.editor()` in the calculator, or with `settings.configMod("dev", "debug", "true")`.

2.1.2 Making a theme

Users can make custom themes for iicalc. This is a guide on how to create them.

Manually making a theme

A theme file has a basic structure. You have the section that tells all of the information about the theme and the section that defines the actual colors. Themes currently support the use of colorama colors and ANSI colors. First, make a new theme file with the `.iitheme` extension. We'll call ours `test.iitheme`. Now, open the theme file in an editor and paste this theme template into it, replacing `NAME` and `DESCRIPTION` with the values that you are using.

```
[theme]
name = NAME
description = DESCRIPTION
ansi = false

[styles]
normal = No
error = No
warning = No
important = No
startupmessage = No
prompt = No
link = No
answer = No
input = No
output = No
```

If you are using ANSI colors, change `ansi` to `true`, if not, leave it as `false`. Now for the fun part, the colors. You can either use ANSI colors, or if you're going for something more basic, you can use colorama. Replace every instance of `No` with the corresponding color that you would like for that value. *Here is a list of what every value is for*. Here are 2 examples, one for colorama and one for ANSI:

Colorama:

```
[styles]
normal = colorama.Fore.RESET + colorama.Back.RESET + colorama.Style.NORMAL
error = colorama.Fore.RED + colorama.Back.RESET + colorama.Style.BRIGHT
warning = colorama.Fore.YELLOW + colorama.Back.RESET + colorama.Style.NORMAL
important = colorama.Fore.MAGENTA + colorama.Back.RESET + colorama.Style.BRIGHT
startupmessage = colorama.Fore.YELLOW + colorama.Back.RESET + colorama.Style.NORMAL
prompt = colorama.Fore.GREEN + colorama.Back.RESET + colorama.Style.BRIGHT
link = colorama.Fore.BLUE + colorama.Back.RESET + colorama.Style.BRIGHT
answer = colorama.Fore.GREEN + colorama.Back.RESET + colorama.Style.NORMAL
input = colorama.Fore.CYAN + colorama.Back.RESET + colorama.Style.NORMAL
output = colorama.Fore.WHITE + colorama.Back.RESET + colorama.Style.NORMAL
```

ANSI:

```
[styles]
normal = \u001b[38;5;238m\u001b[48;5;19m
error = \u001b[38;5;197m\u001b[48;5;201m
warning = \u001b[38;2;219;203;72m
important = \u001b[38;5;236m\u001b[48;5;47m
startupmessage = \u001b[38;5;21m\u001b[48;5;39m
prompt = \u001b[38;5;227m\u001b[48;5;126m
link = \u001b[38;5;26m\u001b[48;5;137m
answer = \u001b[38;5;225m\u001b[48;5;237m
input = \u001b[38;5;178m\u001b[48;5;36m
output = \u001b[38;5;143m\u001b[48;5;225m
```

While ANSI gives more power over the colors, colorama is simpler, so it's up for you to decide which best fits your needs. If you're interested in ANSI, I've found this article fairly useful in getting you started.

Using a tool

If you don't want to manually make a theme, or you want to have more control by using ANSI, but just can't understand that, there are some tools to help you in making themes. I've included a list below of some good ones.

- Builtin `themes.editor()` command - good for making themes from a command line, no ANSI support.
- `themegui` plugin. This plugin can be installed with `pm.install("themegui")` or through `pm.store()` (`store.store()` is now deprecated and will be removed in a future version). This plugin uses Tkinter and has adjustable RGB values with a preview. This plugin also has ANSI support.

2.1.3 Publishing your plugin to the store

CHAPTER

THREE

NOTICE: THE PLUGIN STORE IS NO LONGER AVAILABLE.

3.1 Submiting a plugin

1. Open your browser and go to https://turbowafflz.azurewebsites.net/auth. Note that this make take a minute or two depending if the server is online or not.
2. Click on the "Submit a new plugin" button.
3. **Type:** Click on the first dropdown menu and select if your plugin is a plugin or a theme.
4. **Name:** Go to the name input box, and type your plugin name, preferably a name that is all lowercase.
5. **Description:** Type the description of your plugin or theme in the description box, this will be displayed on the plugin page in the store.
6. **Short Description:** Underneath the description box is the short description, this will be displayed in the store as a sort of preview description.
7. **Version:** Now put in the plugin version, this would probably be `1.0` or `1.0.0` if its the first release version.
8. **Dependencies:** Now you can add dependencies if you are submitting a plugin and not a theme. These can be other plugins, or PyPI packages in a comma separated list. PyPI packages must my prefixed with `pypi:`. For example, if you depended on the `formulas` plugin and the PyPI package `numpy`, you would put `formulas, pypi:numpy`. This can be left blank if there are no dependencies.
9. **Download link:** Go to the download link box. You must use Github or Gitlab to host the files, for instance, I have a central repository for all of my plugins that you can find here. Once you've uploaded your plugin to Github or Gitlab, go to the file, and hit the "Raw" button on the right if on Github, or the "Open Raw" icon button on the right if on Gitlab. Now copy the current URL and paste it into the download link input. It should look something similar to `https://raw.githubusercontent.com/user/repository/branch/file.py` or `https://gitlab.com/user/repository/-/raw/branch/file.py`
10. **File name:** Now, put the filename of the plugin on Github or Gitlab into the filename box.
11. **File hash:** Now you must get the SHA256 hash of the file. Follow the instructions below depending on your operating system. If your OS is not listed, search how to get the SHA256 has of a file on your OS.
    - **Linux**: Navigate to the directory where the file is located in your terminal, and type `sha256sum file.py`, replacing file.py with your filename.
    - **MacOS**: Navigate to the directory where the file is located in your terminal, and type `shasum -a 256 file.py`, replacing file.py with your filename
    - **Windows**: Open PowerShell. Navigate to the directory where the file is located, and type `Get-FileHash file.py | Format-List`, replacing file.py with your filename. Now copy the value after `Hash:`
12. **Calculator Version:** This is where you put the calculator version requirements that are needed for your plugin to run. For example, if your plugin only runs on version 1.5.6 and below, you would type `<=1.5.6` in the text box.

You can add multiple requirements by separating them with commas. To view more about the types of version specification you can do, look at the documentation for PEP 440

13. **Maintainers:** You can add maintainers by their Github usernames. You are added automatically as a maintainer. If you do not press enter after typing a name, it will not be counted as a maintainer, though you can add or remove maintainers later on.

3.2 Updating your plugin

Once you want to update your plugin, you can go to https://turbowafflz.gitlab.io/updateplugin.html or click the "Update an existing plugin" button on the index portal. The same process applies for uploading a plugin, but with something to ease the proccess. If you type your plugin name into the name input and click the "Autofill" button, it will autofill the applicable information. Note that this make take a minute if the server went to sleep. You should increase the version number and recalculate the file hash and make and necessary changes, and then hit the submit button.

3.3 Deleting a plugin

Only the creator of a plugin can delete it. There are 2 ways to do this. The first option is to go here and select it from the list. The second option is to go to here, select your plugin, and then press the delete button on the plugin page. Note that the page's may take a minute to load if the server went to sleep.

3.4 Reporting a plugin

If you find a plugin that violates our *guidelines*, you can report it by going to https://turbowafflz.azurewebsites.net/iicalc/browse, selecting the desired plugin, and pressing the report button on the plugin's page.

3.5 Rating a plugin

There are 2 ways to rate a plugin. The first way is to go to the plugin's page (find it at https://turbowafflz.azurewebsites.net/iicalc/browse) and hit the upvote or downvote button. You must be authenticated to do this. The 2nd way is through the calculator. Start the calculator and run `pm.connect()` if you haven't done that already. Follow the steps and once you're authenticated, you can either run `pm.rate("pluginname")`, or by going to the plugin's store page, accessed with `pm.store()` (`store.store()` is now deprecated and will be removed in a future version).

3.6 Any problems?

Did you get an error or encounter something broken? Start an issue at https://gitlab.com/TurboWafflz/ImaginaryInfinity-Calculator/-/issues

CHAPTER

FOUR

HOW TO ADD PLUGIN SETTINGS TO SETTINGS EDITOR

This is a guide on how to add custom plugin settings to the settings editor. We will use the `discordrpc` plugin as an example.

1. The first step is to add these imports

```
from dialog import Dialog
import configparser
```

We will need `dialog` to display the dialogs and `configparser` to read `config.ini`.

2. The second step is to read the config file. You should put this right underneath your imports so that it gets run on start. You should also set the config variable to global so you can access it everywhere. Just add these lines after your imports.

```
global config
config = configparser.ConfigParser()
config.read("config.ini")
```

This will set a global config variable for your plugin.

3. Now we must make sure that your section in the config is added. This can be whatever you want, but for this example we'll be using a section called `discord`. Underneath the reading of the config, just add:

```
if not config.has_section("discord"):
        config.add_section("discord")
        with open("config.ini", "w") as configFile:
                config.write(configFile)
```

4. The third step is to create your settings class. This will tell the calculator where to look for your settings. Somewhere in your plugin, add a settings class. You will need to put a variable called `choices` in this class. This variable will indicate what goes in the main settings menu, and consists of a bunch of tuples in a list. An example of this is the `Theme` setting, which says `The colors the calculator will use`. Add the settings class like this:

```
class settings:
        choices = [("Discord Rich Presence", "Display ImaginaryInfinity Calculator as␣
↪your status in Discord"), ("Dynamic RPC", "Update Discord RPC on calculation")]
```

As you can see, we're adding 2 settings. 1 entry for `Discord Rich Presence`, and 1 for `Dynamic RPC`.

5. Next we have to add the actual code that handles the setting buttons being pressed. The calculator searches your plugin for a function in the settings class called `settingsPopup`. This function must take 2 arguments, `tag` and `config`. Just add an empty function with this name to your `settings` class. This is what your new settings class should look like:

```
class settings:
        choices = [("Discord Rich Presence", "Display ImaginaryInfinity Calculator as␣
↪your status in Discord"), ("Dynamic RPC", "Update Discord RPC on calculation")]
        def settingsPopup(tag, config):
                pass
```

6. Next, we have to initialize a new dialog in the function.

```
class settings:
        choices = [("Discord Rich Presence", "Display ImaginaryInfinity Calculator as␣
↪your status in Discord"), ("Dynamic RPC", "Update Discord RPC on calculation")]
        def settingsPopup(tag, config):
                d = Dialog(autowidgetsize=True)
```

If we must now check which option the user selected. This value is stored in the `tag` argument. So for each option you have, add a part to the `if` statement.

```
class settings:
        choices = [("Discord Rich Presence", "Display ImaginaryInfinity Calculator as␣
↪your status in Discord"), ("Dynamic RPC", "Update Discord RPC on calculation")]
        def settingsPopup(tag, config):
                d = Dialog(autowidgetsize=True)
                if tag == "Discord Rich Presence":
                        pass
                elif tag == "Dynamic RPC":
                        pass
```

7. Now we must make each value display a dialog menu. Just create a new dialog menu with the different options of the setting in it. `d.menu` takes the arguments `d.menu(menuTitle, choices=[choices])`

```
class settings:
        choices = [("Discord Rich Presence", "Display ImaginaryInfinity Calculator as␣
↪your status in Discord"), ("Dynamic RPC", "Update Discord RPC on calculation")]
        def settingsPopup(tag, config):
                d = Dialog(autowidgetsize=True)
                if tag == "Discord Rich Presence":
                        richPresencMenu = d.menu("Discord Rich Presence", choices=[(
↪"Enable", "Enable Discord RPC"), ("Disable", "Disable Discord RPC")])
                elif tag == "Dynamic RPC":
                        dynamicRPCMenu = d.menu("Update Discord RPC with your last done␣
↪calculation", choices=[("Enable", "Enable Dynamic RPC"), ("Disable", "Disable Dynamic␣
↪RPC")])
```

8. We're almost done now. Now we must check which option of the setting the user selected and apply that change. Just replace `config["discord"]["tagName"]` with `config["mySection"]["myTagName"]`

```
class settings:
        choices = [("Discord Rich Presence", "Display ImaginaryInfinity Calculator as␣
↪your status in Discord"), ("Dynamic RPC", "Update Discord RPC on calculation")]
        def settingsPopup(tag, config):
                d = Dialog(autowidgetsize=True)
                if tag == "Discord Rich Presence":
                        richPresencMenu = d.menu("Discord Rich Presence", choices=[(
```

(continues on next page)

(continued from previous page)

```
↪"Enable", "Enable Discord RPC"), ("Disable", "Disable Discord RPC")])
                        if richPresencMenu[0] == d.OK:
                                # User selected ok instead of cancel
                                if richPresencMenu[1] == "Enable":
                                        config["discord"]["enableRPC"] = "true"
                                else:
                                        config["discord"]["enableRPC"] = "false"
                elif tag == "Dynamic RPC":
                        dynamicRPCMenu = d.menu("Update Discord RPC with your last done␣
↪calculation", choices=[("Enable", "Enable Dynamic RPC"), ("Disable", "Disable Dynamic␣
↪RPC")])
                        if dynamicRPCMenu[0] == d.OK:
                                # User selected ok instead of cancel
                                if dynamicRPCMenu[1] == "Enable":
                                        config["discord"]["dynamicPresence"] = "true"
                                else:
                                        config["discord"]["dynamicPresence"] = "false"
```

9. This last step is ***VERY IMPORTANT***. You ***MUST*** return `config` at the end of the function, or else no settings will be applied. Just add `return config` at the end.

```
class settings:
        choices = [("Discord Rich Presence", "Display ImaginaryInfinity Calculator as␣
↪your status in Discord"), ("Dynamic RPC", "Update Discord RPC on calculation")]
        def settingsPopup(tag, config):
                d = Dialog(autowidgetsize=True)
                if tag == "Discord Rich Presence":
                        richPresencMenu = d.menu("Discord Rich Presence", choices=[(
↪"Enable", "Enable Discord RPC"), ("Disable", "Disable Discord RPC")])
                        if richPresencMenu[1] == "Enable":
                                config["discord"]["enableRPC"] = "true"
                        else:
                                config["discord"]["enableRPC"] = "false"
                elif tag == "Dynamic RPC":
                        dynamicRPCMenu = d.menu("Update Discord RPC with your last done␣
↪calculation", choices=[("Enable", "Enable Dynamic RPC"), ("Disable", "Disable Dynamic␣
↪RPC")])
                        if dynamicRPCMenu[1] == "Enable":
                                config["discord"]["dynamicPresence"] = "true"
                        else:
                                config["discord"]["dynamicPresence"] = "false"
                return config
```

CHAPTER

FIVE

PLUGIN SUBMITTING GUIDELINES

All of the guidelines for submitting plugins can be found here. We reserve the right to remove any plugins from the store at any time for any reason. We decide if you have broken any of the guidelines. By using the plugin store, you agree to these terms.

- Any malicious plugins will result in the plugin being deleted and an immediate ban from submitting plugins.
- The only ads that are allowed are self-promotion ads. You may not receive payment for any ads displayed. Any violation of this rule will result in a warning and the plugin being taken down until it is updated to fit the guidelines.
- Any sexual content will result in a warning and the plugin being taken down until it is updated to fit the guidelines.
- Threats of any kind will not be tolerated and will result in the plugin being deleted and an immediate ban from submitting plugins.
- Revealing someone's personal or confidential information will result in the plugin being deleted and an immediate ban from submitting plugins.
- Illegal content of any kind will not be tolerated and will result in the plugin being deleted and an immediate ban from submitting plugins.
- Promotion of terrorism will not be tolerated and will result in the plugin being deleted and an immediate ban from submitting plugins.
- Purposely spreading false information will result in a warning and the plugin being taken down until it is updated to fit the guidelines.
- Telemetry without consent will result in a warning and the plugin being taken down until it is updated to fit the guidelines.
- search