	Document Title	Function library guide	
	Application Name	Smart Console	
	Document Version	2.0	
	Written By	Vlad Feldfix	Page **1** of **6**

1. Overview

1.1. The Smart-Console is a Python 3 library designed to make console applications for various purposes.

1.2. Smart-Console offers a variety of functions that can save the developer a lot of code writing.

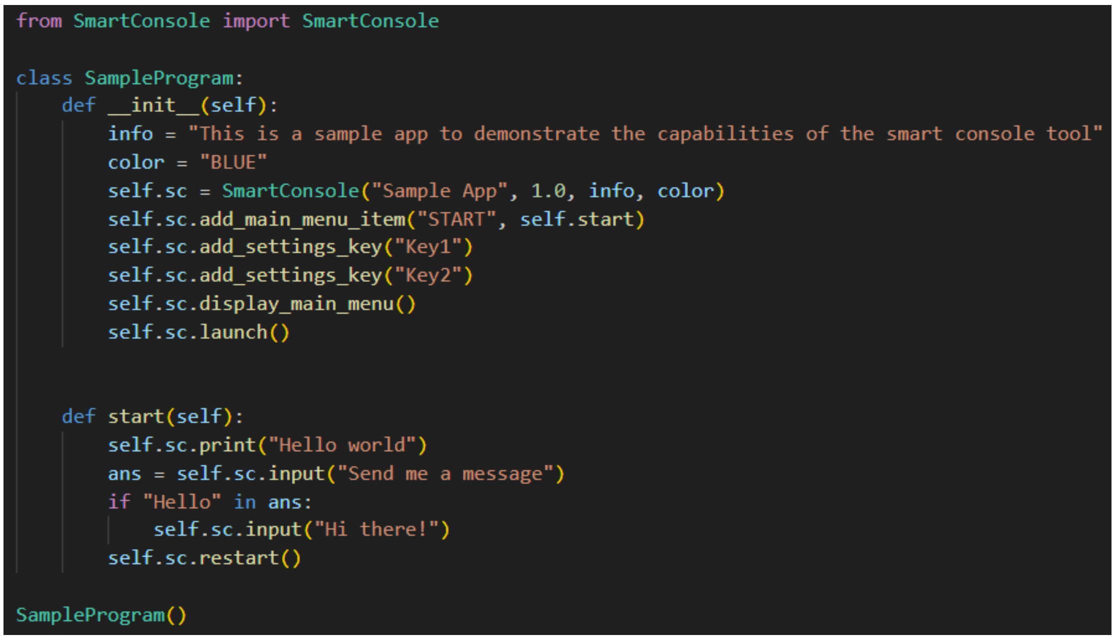
2. How to Use

2.1. Download Smart-Console.py from: https://github.com/VladFeldfix/Smart-Console

2.2. Place the file in your Python folder. (typically: C:\Python\Python313\Lib\SmartConsole.py)

2.3. Once the SmartConsole.py is among your other libraries, you can import it.

2.4. Below is a template for a program using the smart console.



```python
from SmartConsole import SmartConsole

class SampleProgram:
    def __init__(self):
        info = "This is a sample app to demonstrate the capabilities of the smart console tool"
        color = "BLUE"
        self.sc = SmartConsole("Sample App", 1.0, info, color)
        self.sc.add_main_menu_item("START", self.start)
        self.sc.add_settings_key("Key1")
        self.sc.add_settings_key("Key2")
        self.sc.display_main_menu()
        self.sc.launch()


    def start(self):
        self.sc.print("Hello world")
        ans = self.sc.input("Send me a message")
        if "Hello" in ans:
            self.sc.input("Hi there!")
        self.sc.restart()

SampleProgram()
```

Illustration 1 – implementing smart console in your code

2.5. Once Smart-Console is implemented as an object as presented in illustration 1, the developer can use the smart console functions as described in section 3

3. Commands list

3.1. SETUP – the setup functions must be placed in the code in the sequence they are presented in this document.

3.1.1. **__init__ (name, version, info, color):** this is the constructor function and it requires the following arguments:

3.1.1.1. **Name (str)** – the name of your application

3.1.1.2. **Version (any)** – the version of your application

3.1.1.3. **Info (str)** – short description of the application

3.1.1.4. **Color (str)** – the theme color of the app. It can be one of the following options: "RED", "YELLOW", "GREEN", "BLUE", "PURPLE", "PINK", "BROWN", "GREY", or any other color in hex format, for example: "#33cc99"

3.1.2. **add_main_menu_item (function name, function):** the main menu of the application must be set before displayed, and it should contain at least one item. there are several functions that will automatically be added to the end of the list: SETTINGS (if relevant), HELP, VERSION HISTORY, EXIT.

3.1.2.1. **Function name (str)** – the name of the function as displayed in the main menu (for example: "START")

3.1.2.2. **Function (method)** – the function called when user selects this menu item. (for example: self.start)

3.1.3. **add_settings_key (key):** optional function. Adds a key to settings.

3.1.3.1. **Key (str)** – the name of the key. (for example: My files location)

3.1.4. **display_main_menu ():** displays the main menu

3.1.5. **launch ():** starts tkinter's main loop. Must be placed after the setup.

3.1.6. **restart ():** this function will be placed at the end of each main menu function to say "Done!" and display the main menu again.

3.2. MESSAGES

3.2.1. **print (text):** this function displays a messages

3.2.1.1. **Text (str) –** the text of the message to display

3.2.2. **input (text):** this function displays a message and activates the user's entry box allowing the user to type and send a response

3.2.2.1. **Text (str) –** the text of the message to display

3.2.2.2. **RETURN VALUE (str)** – the text of the user's response

3.2.3. **question (text):** this function displays a message with a yes or no question and activates the user's entry box allowing the user to type and send a response

3.2.3.1. **Text (str) –** the text of the message to display

3.2.3.2. **RETURN VALUE (bool)** – True or False based on the user's answer

3.2.4. **choose (text, options):** this function displays a message with a multiple-choice question and activates the user's entry box allowing the user to type and send a response

3.2.4.1. **Text (str) –** the text of the message to display

3.2.4.2. **Options (tuple) –** a list of choices to select

3.2.4.3. **RETURN VALUE (str)** – the option the user selected

3.3. SETTINGS

3.3.1. **get_settings_value (key):** this function returns the current value of the requested key from settings.

3.3.1.1. **Key (str) –** the requested key

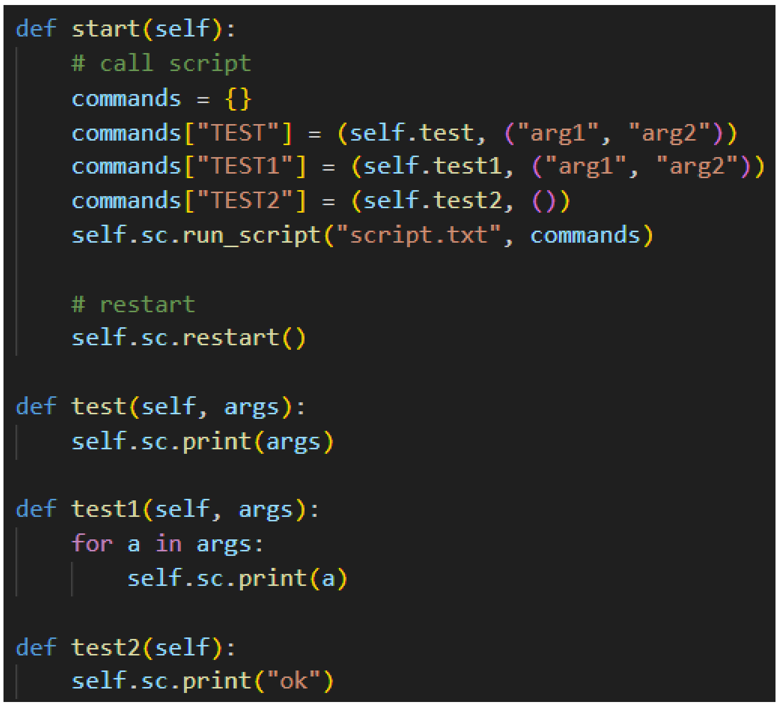
3.4. SCRIPT

3.4.1. **run_script (path, functions):** runs a script with the given functions. See Illustration 2 implementation example.

3.4.1.1. **Path (str) –** the location of the script file (will display a message saying the file is not found if that's the case)

3.4.1.2. **Functions (dict) –** a dictionary of functions and expected arguments. The structure of the dict must be as following:
key: FUNCTION NAME (str)
value: (FUNCTION TO CALL (method), LIST OF ARGUMENTS (tuple))
For example:
{ "SAY HELLO": (self.say_hello, ("argument1", " argument2")) }

3.4.1.3. **RETURN VALUE (bool) –** True or False whether the code went successfully without internal errors



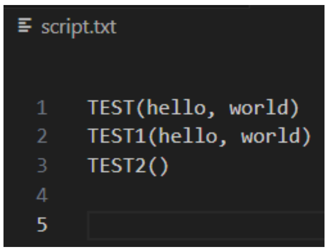
```
def start(self):
    # call script
    commands = {}
    commands["TEST"] = (self.test, ("arg1", "arg2"))
    commands["TEST1"] = (self.test1, ("arg1", "arg2"))
    commands["TEST2"] = (self.test2, ())
    self.sc.run_script("script.txt", commands)

    # restart
    self.sc.restart()

def test(self, args):
    self.sc.print(args)

def test1(self, args):
    for a in args:
        self.sc.print(a)

def test2(self):
    self.sc.print("ok")
```



script.txt

```
TEST(hello, world)
TEST1(hello, world)
TEST2()
```

Illustration 2 – run script implementation example

3.5. DATABASE

3.5.1. **database_connect(name, path, headers):** connect to a database csv file

3.5.1.1. **Name (str) –** is the name of the database (e.g. "My Contacts List")

3.5.1.2. Path (str) – the location of the database file (e.g. "C:/Users/My Documents")

3.5.1.3. **Headers (tuple) –** a tuple of database headers (e.g. ("First Name", "Last Name", "Phone Number")).

3.5.2. **database_insert(name, data):** insert a new line to a database

3.5.2.1. **Name (str) –** the name of the database

3.5.2.2. **Data (tuple) –** a line to insert (e.g. ("Vlad", "Feldfix", "12547856")).

3.5.3. **database_delete(name, key):** deletes a primary key from a database.

3.5.3.1. **Name (str) –** the name of the database

3.5.3.2. **Key (str) –** the PK to delete (e.g. "Vlad")

3.5.4. **database_commit(name):** save database to file. Without calling this function the database will not be saved to the file.

3.5.4.1. **Name (str)** - the name of the database

3.5.5. **database_data(name):** returns all the data from a csv file

3.5.5.1. **Name (str) -** the name of the database

3.5.5.2. **RETURN VALUE (dict.items()) –** a list of key, value items. Where KEY is the primary key and VALUE is the rest of the line

3.5.6. **database_headers(name):** returns the headers of the database

3.5.6.1. **Name (str) -** the name of the database

3.5.6.2. **RETURN VALUE (tuple) –** a tuple of the headers of the database. (e.g. ("First Name", "Last Name", "Phone Number")).

3.6. DATE

3.6.1. **today():** return the current date in format YYYY-MM-DD

3.6.1.1. **RETURN VALUE (str) –** YYYY-MM-DD

3.6.2. **now():** return the current time in format HH:MM:SS

3.6.2.1. **RETURN VALUE (str) –** HH:MM:SS

3.6.3. **current_year():** return the current year in YYYY

3.6.3.1. **RETURN VALUE (str) –** YYYY

3.6.4. **current_month():** return the current month in MM

3.6.4.1. **RETURN VALUE (str) –** MM

3.6.5. **current_day():** return the current day in DD

3.6.5.1. **RETURN VALUE (str) –** DD

3.6.6. **current_hour():** return the current hour in HH

3.6.6.1. **RETURN VALUE (str) –** HH

3.6.7. **current_minute():** return the current minute in MM

3.6.7.1. **RETURN VALUE (str) –** MM

3.6.8. **current_second():** return the current second in SS

3.6.8.1. **RETURN VALUE (str) –** SS

3.6.9. **compare_dates(date1, date2):** compare 2 given dates.

3.6.9.1. **Date1 (str) –** must be given in format "YYYY-MM-DD"

3.6.9.2. **Date2 (str) –** must be given in format "YYYY-MM-DD"

3.6.9.3. **RETURN VALUE (int) –** returns the number of days between the two dates.

If DATE 1 is before DATE 2 the return value is positive

If DATE 1 is after DATE 2 the return value is negative

If DATE 1 is the same as DATE 2 the return value is zero

3.7. LOG

3.7.1. **write_to_log(text):** adds a new line to the log

3.7.1.1. **Text (str) –** the content of the line

3.7.2. **add_log_header(text):** adds a new header the log. The default headers are: Date: YYYY-MM-DD HH:MM:SS, and a list of settings.

3.7.2.1. **Text (str) –** the text of the header. (e.g. "Tested files: 758")

3.7.3. **display_log():** displays the log as an html file.

3.8. FILES

3.8.1. **test_paths(paths):** make sure that all paths are good

3.8.1.1. **Paths (tuple) –** a tuple of paths to test

3.8.1.2. **RETURN VALUE (bool) –** True or False if all the paths are good or not. (good is a folder or a file)

3.8.2. **open(path):** opens a file or folder in explorer

3.8.2.1. **path (txt) –** the file or folder to open

4. Document version history

4.1. v1.0 created on 2025-03-17

4.2. v2.0 created on 2025-10-20