How to Encode and Decode Messages Using Base64 and Python

This simple App GUI is a utility tool. Developing it will help you hone your Python skills.

Python's Base64 module is a powerful tool for encoding and decoding messages. You can use it to send data securely over the Internet. It's standard practice for web, apps, and communication services to use this type of encryption to protect sensitive data from malicious hackers.

The Base64 module has a pair of functions that you can use to encode and decode messages, increasing security when transferring data.

Tkinter and Base64 . Modules

Tkinter allows you to create desktop applications. It provides a wide range of widgets like buttons, stickers and text boxes for you to easily develop your app without much effort. You can create impressive GUI programs with Tkinter. You can build a simple calculator, to-do list app or a game that tests your typing skills. To install Tkinter on your system, open a terminal and type:

```
pip install tkinter
```

BASE64 provides functions to encode binary data so that ASCII decodes them back to binary. It supports both standard encryption and secure URLs. That makes the transfer of information more secure. To convert a string to a Base64 character, take the ASCII value of each character and calculate its 8-bit binary value. Convert this 6-bit chunk by grouping the numbers and converting them back to their respective decimal values. Finally, use the

Base64 encoding to get the Base64 value for each decimal.

How to Encrypt and Decrypt Messages in Python

Start by importing both modules. Initialize Tkinter instance and show root window. Set the title, pixel size, and background color of the window.

```
from tkinter import * import base64 root = Tk() root.geometry('750x400') root.co
```

Use the Label widget to display useful information about this application. This widget accepts a main window in which you want to place it, displaying the content, font style, color and background color. Use **pack()** to arrange the widget in a layout block before placing it in the main widget. StringVar makes it easier to control the widget's value, such as through Label or Entry.

```
Label(root, text='Python Message Encoder and Decoder', font='arial 25 bold', fg=
```

Define an **Encode()** function that accepts a key for encryption and decryption along with the message. Define an empty list and iterate over the length of the message. Set the index of the key as the active module and store its value in the variable **key_c** . Use **ord()** to get the Unicode value of the character and use **chr()** to get the wildcard for the specified value.

Add this value to the list. Append each element of the list to an empty string and use the encode method to return a utf-8 encoded version of this string. **The base64.urlsafe_b64encode()** method encodes this input field and replaces **-** with **+** and **_** with **/** .

```
def Encode(key, message): ????enc = [] ????for i in range(len(message)):
????????key_c = key[i % len(key)] ????????
enc.append(chr((ord(message[i]) + ord(key_c)) % 256)) ????
return base64.urlsafe_b64encode("".join(enc).encode()).decode()
```

Define a **Decode()** function that accepts a key for encryption and decryption along with the message. Define an empty list and decode the message. Iterate over the length of the message and set the active module as the index and store its value in **key_c** . Add Unicode string message decoding character as shown below. Returns the decoded string.

```
def Decode(key, message): ????dec = [] ????
message = base64.urlsafe_b64decode(message).decode() ????
for i in range(len(message)): ????????key_c = key[i % len(key)] ????????
dec.append(chr((256 + ord(message[i]) - ord(key_c)) % 256)) ????
return "".join(dec)
```

Define a **Mode()** function that takes the mode entered by the user in the Entry widget and calls the appropriate function according to the selection. In case the user does not enter a valid response, an error message is displayed.

```
def Mode(): ????if (mode.get() == 'E'): ????????
Result.set(Encode(key.get(), Text.get())) ????elif (mode.get() == 'D'):
????????Result.set(Decode(key.get(), Text.get())) ????else: ????????
Result.set('Invalid Mode')
```

Define an **Exit()** function to shut down and end the interpreter running in the background. Define a **Reset()** function to remove the contents of the Entry field.

```
def Exit(): ????root.destroy() def Reset(): ????Text.set("") ????
key.set("") ????mode.set("") ????Result.set("")
```

Define Label and Entry widgets for Message, Key, Mode and Text. Specify the main window you want to place them inside, including the font style, text, and background color. Also, set the coordinates to align them properly.

```
Label(root, font='arial 17 bold', text='Message', fg='black', bg="aqua").place(x=
```

Similarly, define 3 buttons to view results, reset fields and pause the program. Buttons with special parameters are called commands that take a function and deploy it when clicked.

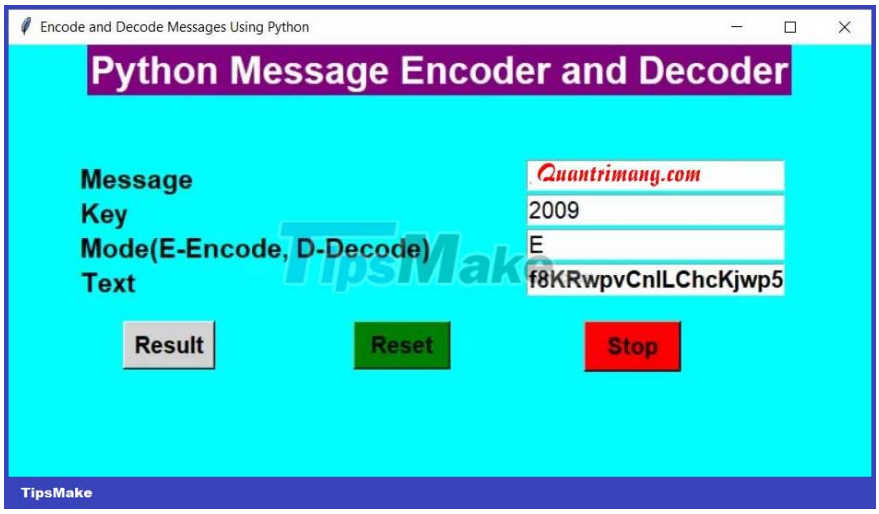
```
Button(root, font='arial 15 bold', text='Result', padx=2, bg='Light Gray', comma
```

The mainloop() function calls Python to loop through the Tkinter event and listen for events (such as button clicks) until you close the window.

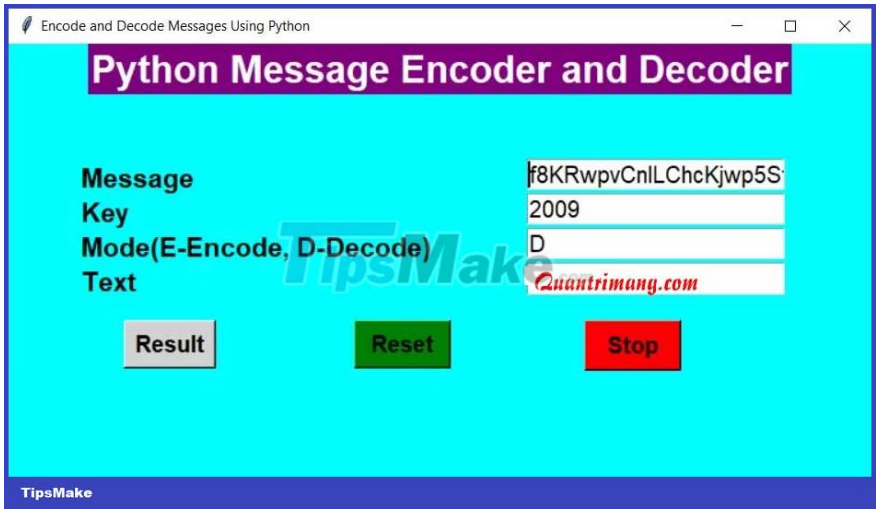
```
root.mainloop()
```

Encrypt/Decrypt message on action

When running this program, it shows a window where you have to enter a message, lock and mode. When selecting Encoding mode with a 2009 key, the message content will turn into **f8KRwpvCnlLChcKjwp5Sf8KW**
.



Similarly, when you copy the encrypted message and paste it back into the input field for decryption, you get the original message.



Above is **how to encode and decode messages in Python** . Hope the article is useful to you.

You finished reading the article "**How to Encode and Decode Messages Using Base64 and Python**" edited by the **TipsMake** team. We hope this article has provided you with many useful tech tips and tricks. You can search for similar articles on tips and guides. Thank you for reading and for following us regularly.
