

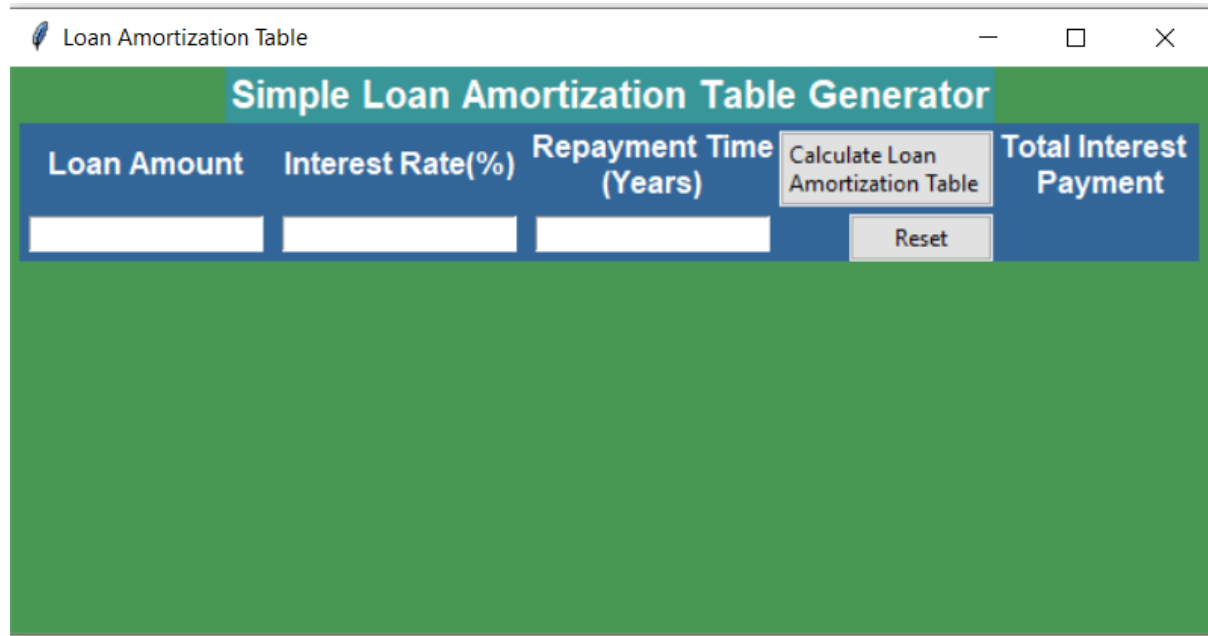
Simple Loan Amortization Table Generator

---Abhishek Sarkar

Language Used: Python

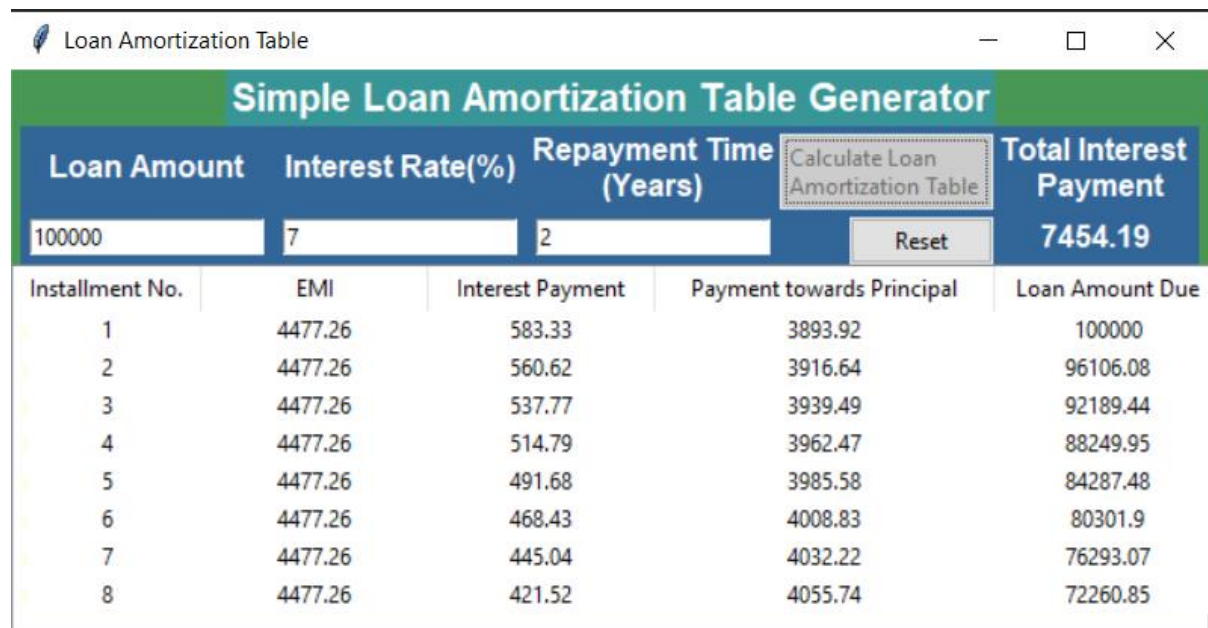
GUI Tool: Tkinter

Initial Output: -



The screenshot shows a Tkinter window titled "Loan Amortization Table". The window has a green header bar with the title "Simple Loan Amortization Table Generator". Below the header, there is a blue bar containing the input fields and buttons. The input fields are labeled "Loan Amount", "Interest Rate(%)", and "Repayment Time (Years)". To the right of these fields are two buttons: "Calculate Loan Amortization Table" and "Reset". To the far right of the blue bar is a label "Total Interest Payment". The main area of the window is green and currently empty.

Output with Values: -



The screenshot shows the same Tkinter window as before, but now with values entered in the input fields: "Loan Amount" is 100000, "Interest Rate(%)" is 7, and "Repayment Time (Years)" is 2. The "Calculate Loan Amortization Table" button is highlighted with a dashed border. The "Total Interest Payment" label now shows the value 7454.19. Below the blue bar, a table is displayed with the following data:

Installment No.	EMI	Interest Payment	Payment towards Principal	Loan Amount Due
1	4477.26	583.33	3893.92	100000
2	4477.26	560.62	3916.64	96106.08
3	4477.26	537.77	3939.49	92189.44
4	4477.26	514.79	3962.47	88249.95
5	4477.26	491.68	3985.58	84287.48
6	4477.26	468.43	4008.83	80301.9
7	4477.26	445.04	4032.22	76293.07
8	4477.26	421.52	4055.74	72260.85

Codes:

```
import tkinter as tk
from tkinter import *
from tkinter import messagebox
from tkinter import ttk

root = Tk()
root.title("Loan Amortization Table")
root.geometry("635x300")
root.configure(bg="#489754")
fr3 = tk.Frame(root)
fr3.pack(expand=NO, fill='y')
fr2 = tk.Frame(root, background="#326598")
fr2.pack(expand=NO, fill='y')
fr1 = tk.Frame(root, background="#489754")
fr1.pack(expand=YES, fill=BOTH)

welcome = tk.Label(fr3, text="Simple Loan Amortization Table Generator",
font=("Cooper Std", 15, "bold"), fg="White",
background="#389598")
welcome.pack()

var1 = tk.StringVar(fr2)
var2 = tk.StringVar(fr2)
var3 = tk.StringVar(fr2)

amount = tk.Label(fr2, text="Loan Amount", font=("Cooper Std", 12, "bold"),
fg="White", background="#326598")
amount.grid(row=0, column=0, ipadx=5, ipady=5)
e1 = tk.Entry(fr2, textvariable=var1)
e1.grid(row=1, column=0, padx=5, pady=5)
Rate = tk.Label(fr2, text="Interest Rate(%)", font=("Cooper Std", 12,
"bold"), fg="White", background="#326598")
Rate.grid(row=0, column=1)
e2 = tk.Entry(fr2, textvariable=var2)
e2.grid(row=1, column=1, padx=5, pady=5)
tenur = tk.Label(fr2, text="Repayment Time\n(Years)", font=("Cooper Std",
12, "bold"), fg="White", background="#326598")
tenur.grid(row=0, column=2)
e3 = tk.Entry(fr2, textvariable=var3)
e3.grid(row=1, column=2, padx=5, pady=5)
totpay = tk.Label(fr2, text="Total Interest \n Payment", font=("Cooper
Std", 12, "bold"), fg="White",
background="#326598")
totpay.grid(row=0, column=4)

def data_entry(event):
    var4 = (var1.get())
    var5 = (var2.get())
    var6 = (var3.get())
    try:
        int(var4) and float(var5) and int(var6)
        a = int(var4)
        b = float(var5)
        c = int(var6)
        m = (a * (b / 1200)) * (((1 + (b / 1200)) ** (c * 12)) / (((1 + (b
/ 1200)) ** (c * 12))) - 1))
        f = c * 12

        p = [a, ]
```

```

d = []
h = []
for i in p:
    p.append(round((i - (m - (i * (b / 1200))))), 2))
    d.append(round((i * (b / 1200))), 2))
    h.append(round((m - (i * (b / 1200))), 2))
    if len(p) <= f:
        pass
    else:
        break

g = p[0:-1]
o = range(1, ((c * 12) + 1))

data = list(zip(g, d, h, o))

global my_tree

my_tree = ttk.Treeview(fr1, selectmode="extended")
my_tree['columns'] = (
    "Installment No.", "EMI", "Interest Payment", "Payment towards
Principal", "Loan Amount Due")

my_tree.column("#0", anchor=CENTER, width=0, stretch=NO)
my_tree.column("Installment No.", anchor=CENTER, width=100)
my_tree.column("EMI", anchor=CENTER, width=120)
my_tree.column("Interest Payment", anchor=CENTER, width=120)
my_tree.column("Payment towards Principal", anchor=CENTER,
width=180)
my_tree.column("Loan Amount Due", anchor=CENTER, width=120)

my_tree.heading("#0", text="", anchor=CENTER)
my_tree.heading("Installment No.", text="Installment No.",
anchor=CENTER)
my_tree.heading("EMI", text="EMI", anchor=CENTER)
my_tree.heading("Interest Payment", text="Interest Payment",
anchor=CENTER)
my_tree.heading("Payment towards Principal", text="Payment towards
Principal", anchor=CENTER)
my_tree.heading("Loan Amount Due", text="Loan Amount Due",
anchor=CENTER)

count = 0
for record in data:
    my_tree.insert(parent='', index='end', iid=count, text='',
values=(record[3], round(m, 2), record[1],
record[2], record[0]))
    count += 1

my_tree.pack(expand=YES, fill=BOTH)
global totpay1
totpay1 = tk.Label(fr2, text=str(round(((m * f) - a), 2)),
font=("Cooper Std", 12, "bold"), fg="White",
background="#326598")
totpay1.grid(row=1, column=4)
b1.config(state="disabled")

except:
    tk.messagebox.showerror("Error in Input Data",
        "Please Enter proper Numbers,\nDecimal only
allowed for Interest rate")

```

```

b1 = ttk.Button(fr2, text="Calculate Loan \nAmortization Table ")
b1.bind("<Button-1>", data_entry)
b1.bind("<Return>", data_entry)
b1.grid(row=0, column=3, sticky='se')

def reset(event):
    totpay1.destroy()
    my_tree.destroy()

    b1.config(state="enabled")

b2 = ttk.Button(fr2, text="Reset")
b2.bind("<Button-1>", reset)
b2.bind("<Return>", reset)
b2.grid(row=1, column=3, sticky='se')

root.mainloop()

```

Codes Breakdown:

```

import tkinter as tk
from tkinter import *
from tkinter import messagebox
from tkinter import ttk

```

Importing the GUI module along with widgets.

```
root = Tk()
```

Setting up the Initial window as root.

```
root.title("Loan Amortization Table")
```

Setting up the name.

```

root.geometry("635x300")
root.configure(bg="#489754")

```

Setting up window size and background colour.

```

fr3 = tk.Frame(root)
fr3.pack(expand=NO, fill='y')
fr2 = tk.Frame(root, background="#326598")
fr2.pack(expand=NO, fill='y')
fr1 = tk.Frame(root, background="#489754")
fr1.pack(expand=YES, fill=BOTH)

```

Making three different frame for three different purpose, Title, Input and Output. All directed to “root” and “pack” is used to set the geometry. Argument “Expand” is used to determine that if user expands the window which frame to expand which not to expand. Fill argument is used to determine in which direction the frame should expand. “fr1” will expand in both direction unlike the others.

```

welcome = tk.Label(fr3, text="Simple Loan Amortization Table Generator",
font=("Cooper Std", 15, "bold"), fg="White",
background="#389598")
welcome.pack()

```

Welcome Title along with necessary arguments using “Label” widget in Tkinter.

```
var1 = tk.StringVar(fr2)
var2 = tk.StringVar(fr2)
var3 = tk.StringVar(fr2)
```

Creating three variable for Inputs. And “StringVar” is used to get the Containing text of the Input entries.

```
amount = tk.Label(fr2, text="Loan Amount", font=("Cooper Std", 12, "bold"),
fg="White", background="#326598")
amount.grid(row=0, column=0, padx=5, pady=5)
```

Use of Label to name a input entry. Since this comes under Input section, it is directed to the “fr2” and using “Grid” option the position is marked. The frame becomes like a Table where we define the position of each object.

```
e1 = tk.Entry(fr2, textvariable=var1)
e1.grid(row=1, column=0, padx=5, pady=5)
```

Using “Entry” widget of the Tkinter we have made the first entry field that will collect the Total Loan amount.

```
Rate = tk.Label(fr2, text="Interest Rate(%)", font=("Cooper Std", 12,
"bold"), fg="White", background="#326598")
Rate.grid(row=0, column=1)
e2 = tk.Entry(fr2, textvariable=var2)
e2.grid(row=1, column=1, padx=5, pady=5)
tenur = tk.Label(fr2, text="Repayment Time\n(Years)", font=("Cooper Std",
12, "bold"), fg="White", background="#326598")
tenur.grid(row=0, column=2)
e3 = tk.Entry(fr2, textvariable=var3)
e3.grid(row=1, column=2, padx=5, pady=5)
totpay = tk.Label(fr2, text="Total Interest \n Payment", font=("Cooper
Std", 12, "bold"), fg="White",
background="#326598")
totpay.grid(row=0, column=4)
```

Similar codes related to previous description.

```
def data_entry(event):
```

A function “data_entry” is created to execute the main calculation. “event” is passed through the function to bind it in future.

```
var4 = (var1.get())
var5 = (var2.get())
var6 = (var3.get())
```

Three more variable are created to get the values from var1, var2 and var3. We used StringVar there just to get the values using “get()” here.

```
try:
    int(var4) and float(var5) and int(var6)
```

Logical check of the values collected from user in the input fields. Only Decimal String is accepted in Interest rate field So we use float for the second field. “int” will check for integer.

```
a = int(var4)
b = float(var5)
c = int(var6)
```

“a” collects the loan amount, “b” collects the Interest amount and “c” collects the No. of year needed for Repayment.

```
m = (a * (b / 1200)) * (((1 + (b / 1200)) ** (c * 12)) / (((1 + (b / 1200)) ** (c * 12)) - 1))
```

EMI calculation is done in this step and stored in “m”. Mathematical formula is used:
 $EMI = P \times r \times (1 + r)^n / ((1 + r)^n - 1)$ where P= Loan amount, r= interest rate, n=tenure in number of months.

```
p = [a, ]
d = []
h = []
```

Three lists are created to hold the recurrent values generated in the following for loop. “p” holds the recurrent Balance starting with the capital as Balance.

```
for i in p:
    p.append(round((i - (m - (i * (b / 1200)))), 2))
    d.append(round((i * (b / 1200)), 2))
    h.append(round((m - (i * (b / 1200))), 2))
```

For loop which will be adding values to the lists using the “append” command. All the values are rounded to two decimal points. Payment towards Interest is stored in the list “d” and payment towards Principal is stored in “h”. Mathematical formula for calculation of Payment towards Interest is : = Amount Due x (rate of interest/12) .

“i” will start with the “a” value that is the initial due and the first value of the list “p”. Then with append command successive dues will be generated and will be stored in the list. Similarly same process will go on for “d” and “h”.

```
if len(p) <= f:
    pass
else:
    break
```

Since there is no upper limit in “p” the “for” loop will be carried till infinity that’s why we need to cap it with a “if” clause.

```
f = c * 12
```

This is used to get the number of EMI and its just calculated the year with 12. As long as the length of “p” is less than or equal to the number of EMI the for loop will go on, else it will break.

```
g = p[0:-1]
o = range(1, ((c * 12) + 1))
```

If the process goes on “p” will have one extra value than “d” and “h” since it starts with one extra value. And the last value will be a negative value. That’s why another list is formed to avoid this problem which is “g”.

```
data = list(zip(g, d, h, o))
```

Comma separated data base is created using the “zip” command.

```
global my_tree

my_tree = ttk.Treeview(frl, selectmode="extended")
my_tree['columns'] = (
    "Installment No.", "EMI", "Interest Payment", "Payment towards
    Principal", "Loan Amount Due")
```

A global variable “my_tree” is created since it will also be used in another function. “TreeView” widget is used make the table. It is put in the third frame or the output

frame that we have created naming “fr1”. Columns are added using the necessary argument.

```
my_tree.column("#0", anchor=CENTER, width=0, stretch=NO)
my_tree.column("Installment No.", anchor=CENTER, width=100)
my_tree.column("EMI", anchor=CENTER, width=120)
my_tree.column("Interest Payment", anchor=CENTER, width=120)
my_tree.column("Payment towards Principal", anchor=CENTER, width=180)
my_tree.column("Loan Amount Due", anchor=CENTER, width=120)
```

Properties of Columns are set using these commands. The first column is default of TreeView and to hide that we made the “stretch” argument as “NO”.

```
my_tree.heading("#0", text="", anchor=CENTER)
my_tree.heading("Installment No.", text="Installment No.", anchor=CENTER)
my_tree.heading("EMI", text="EMI", anchor=CENTER)
my_tree.heading("Interest Payment", text="Interest Payment", anchor=CENTER)
my_tree.heading("Payment towards Principal", text="Payment towards Principal", anchor=CENTER)
my_tree.heading("Loan Amount Due", text="Loan Amount Due", anchor=CENTER)
```

Heading of columns are given in these codes and aligned with “anchor” argument.

```
count = 0
for record in data:
    my_tree.insert(parent='', index='end', iid=count, text='',
                  values=(record[3], round(m, 2), record[1], record[2],
record[0]))
    count += 1
```

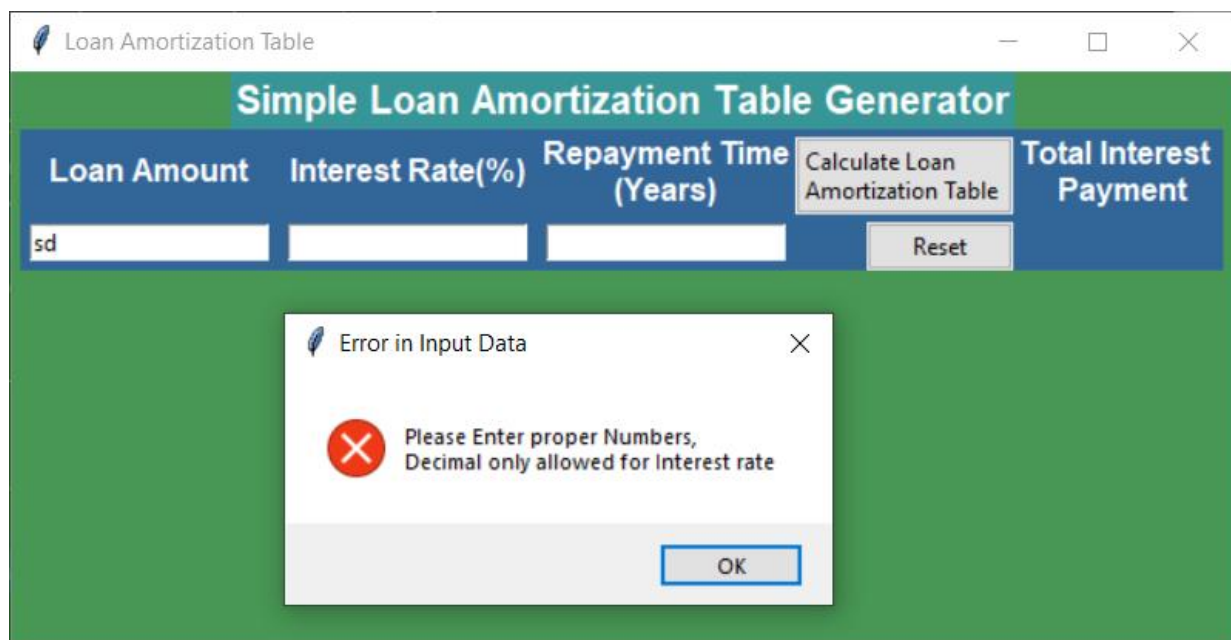
Data inserted in the TreeView using the for loop and the data set called “data”. ‘parent’ is left blank since we have hidden that. Values of the data set “data” is called one by one according to the name of column.

```
my_tree.pack(expand=YES, fill=BOTH)
```

Allocation and packing of the widget.

```
global totpay1
totpay1 = tk.Label(fr2, text=str(round(((m * f) - a), 2)), font=("Cooper Std", 12, "bold"), fg="White",
                  background="#326598")
totpay1.grid(row=1, column=4)
```

This is used to add value to the fifth label that we have created at the very beginning. The value is the total interest payment and it is displayed there in the stipulated area.



```
b1.config(state="disabled")
```

b1 is a button that we will discuss later.

```
except:
    tk.messagebox.showerror("Error in Input Data",
                            "Please Enter proper Numbers,\nDecimal only
allowed for Interest rate")
```

We started the whole Calculation with a Logical test using “try” at the beginning if it passes the test the calculation starts otherwise with “except” argument a message box will popup showing the error message.

```
b1 = ttk.Button(fr2, text="Calculate Loan \nAmortization Table ")
b1.bind("<Button-1>", data_entry)
b1.bind("<Return>", data_entry)
b1.grid(row=0, column=3, sticky='se')
```

The first button created using Tkinter widget “Button” , Placed in input frame that is “fr2”, named accordingly. Pair of “bind” is used to make the Button working with Mouse Click as well as Enter key in the keyboard. Since we have to bind the button we haven’t used the “command” argument instead we have called the function “data_enty” in both the binds. Using “grid” the button is positioned properly.

```
b1.config(state="disabled")
```

This line of code is written earlier, to just disable the Button once the calculation is done and output is generated till further intimation.

```
def reset(event):
    totpay1.destroy()
    my_tree.destroy()

    b1.config(state="enabled")
```

Another function is created to that will reset to the initial window. “destroy” argument will just destroy the two outputs. And along with that will reactive the Button “b1”.

```
b2 = ttk.Button(fr2, text="Reset")
b2.bind("<Button-1>", reset)
b2.bind("<Return>", reset)
b2.grid(row=1, column=3, sticky='se')
```

Second Button as “b2” is placed similarly.

```
root.mainloop()
```

“root” that is the main window is kept activated with this code.

Using “pyinstaller” the “.py” file is converted into a executable file. And it is good to go in any Latest windows computer.

How to Run:

Just Double Click the .exe file it will run in any modern desktop.

©Abhishek Sarkar

14/03/2021

7278184078

Abhishek.kol28@gmail.com