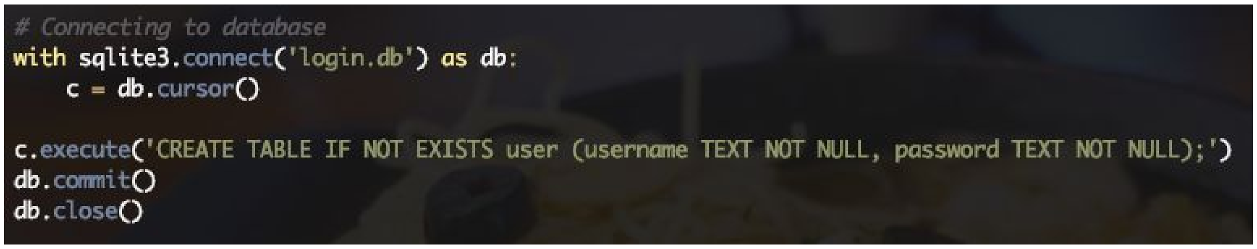
Login_file (login.py)

- ❏ SQLite3 database and its commands
- ❏ Each method as a function in a class

Household_account file (IA.py)

- ❏ Each window is displayed through a single loop and a class.
- ❏ A class for a window
- ❏ Key event
- ❏ A graph

- **Login file -**
  Connecting to a database and creating a table



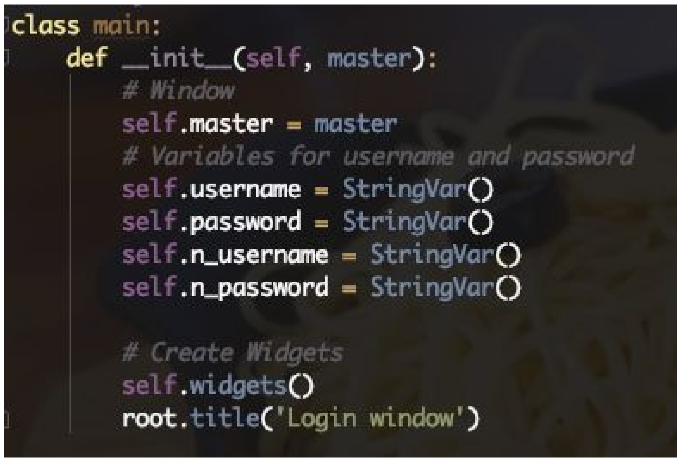
```
# Connecting to database
with sqlite3.connect('login.db') as db:
    c = db.cursor()

c.execute('CREATE TABLE IF NOT EXISTS user (username TEXT NOT NULL, password TEXT NOT NULL);')
db.commit()
db.close()
```

By importing sqlite3, `import sqlite3`, the program connects to a database called `login.db` and `c` is defined as `db.cursor()`. The cursor is a control structure that enables traversal over the records in a database that processes with the traversal retrieval or addition of records, and is viewed as a pointer to one row in a set of rows.

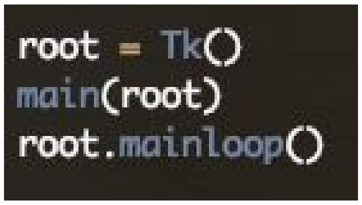
A “user” table is created in a database with columns of username and password with both TEXT required, NOT NULL.

`commit()` updates a record to make changes so the database commits and closes.

Setting of the program (username and password):



```
class main:
    def __init__(self, master):
        # Window
        self.master = master
        # Variables for username and password
        self.username = StringVar()
        self.password = StringVar()
        self.n_username = StringVar()
        self.n_password = StringVar()

        # Create Widgets
        self.widgets()
        root.title('Login window')
```



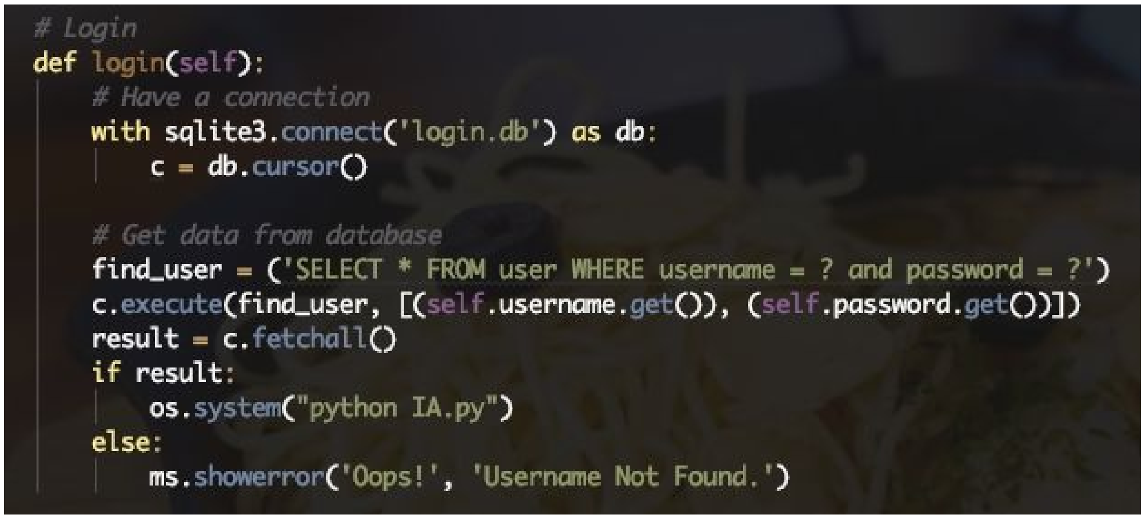
```
root = Tk()
main(root)
root.mainloop()
```

This is the main class and has a built-in `__init__(self, master)` function and always executed when the class is initiated. This function is used to assign values to object properties or other operations that are necessary when the object is created.

For example, `self.master = master` or `self.username = StringVar()` are assigned values. Those objects are variables for username and password, and `self.widgets()` creates widgets.

This class is executed as a variable `root` defines `Tk()` in the `main` class as `main(root)`, and has `root.mainloop()` at the end to halt the program.

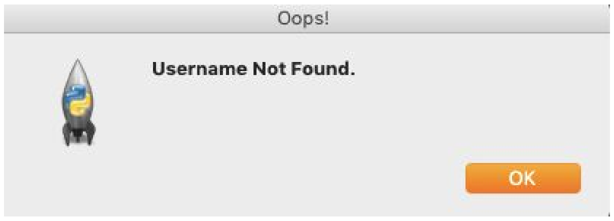
Login function:



```
# Login
def login(self):
    # Have a connection
    with sqlite3.connect('login.db') as db:
        c = db.cursor()

    # Get data from database
    find_user = ('SELECT * FROM user WHERE username = ? and password = ?')
    c.execute(find_user, [(self.username.get()), (self.password.get())])
    result = c.fetchall()
    if result:
        os.system("python IA.py")
    else:
        ms.showerror('Oops!', 'Username Not Found.')
```

Objects can contain methods, which are functions that belong to the object. This is `login` method that allows the user to login. First it connects to the database `login.db` and finds user information from the table, `user`, where the `username` and `password` are placeholders denoted by `?`. Then it grabs input `username` and `password` in the entries by the `user`, and matches if they are correct through executing and fetching.

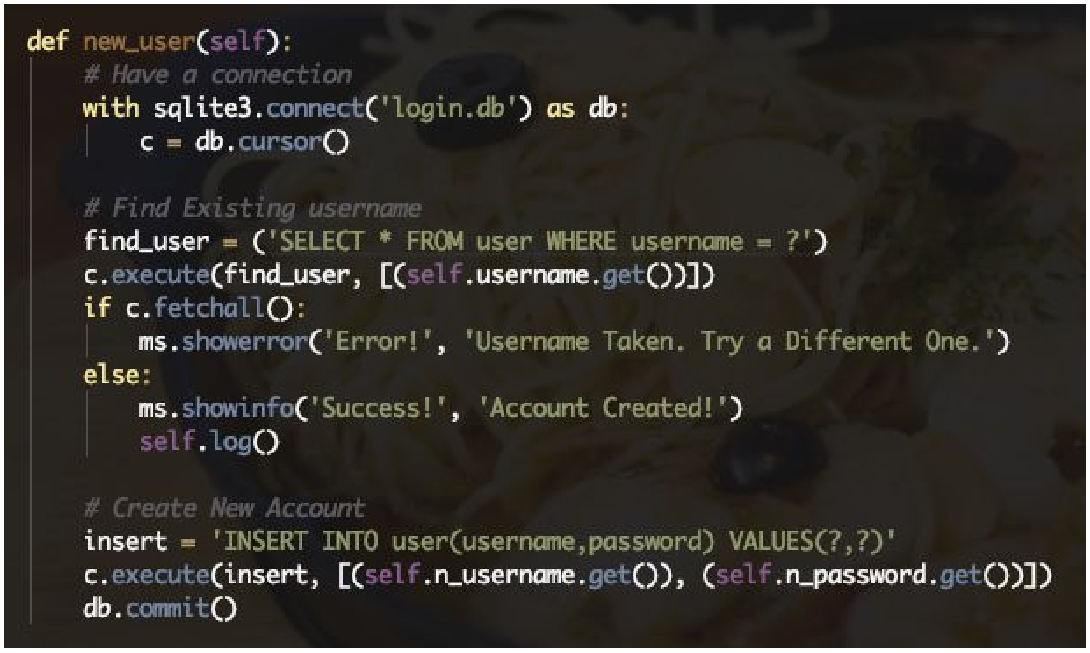
If the fetching, defined as `result`, is correct, it will open up a new python file (`IA.py`), household account. If not, it shows an error message through a messagebox (`ms`).



Oops!

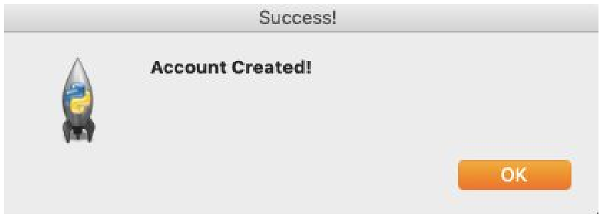
Username Not Found.

Creating a new account:



```
def new_user(self):
    # Have a connection
    with sqlite3.connect('login.db') as db:
        c = db.cursor()

    # Find Existing username
    find_user = ('SELECT * FROM user WHERE username = ?')
    c.execute(find_user, [(self.username.get())])
    if c.fetchall():
        ms.showerror('Error!', 'Username Taken. Try a Different One.')
    else:
        ms.showinfo('Success!', 'Account Created!')
        self.log()

    # Create New Account
    insert = 'INSERT INTO user(username,password) VALUES(?,?)'
    c.execute(insert, [(self.n_username.get()), (self.n_password.get())])
    db.commit()
```

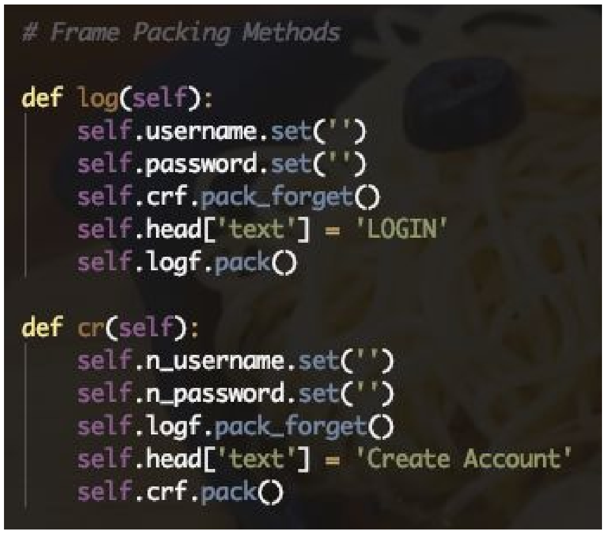
This is the same process as above, but it creates an account. It connects to the database and checks if entered username is taken or not. If taken, it shows a success message.



Success!

Account Created!

Functions of logging in and creating:

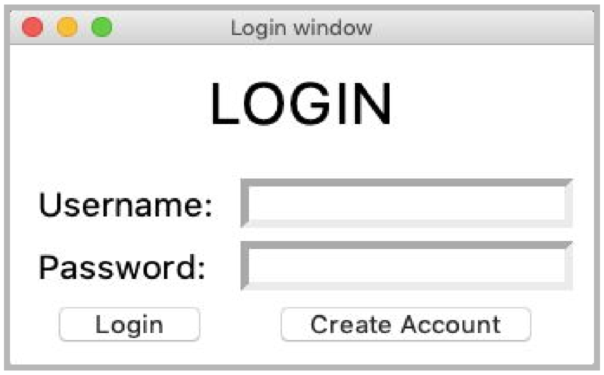


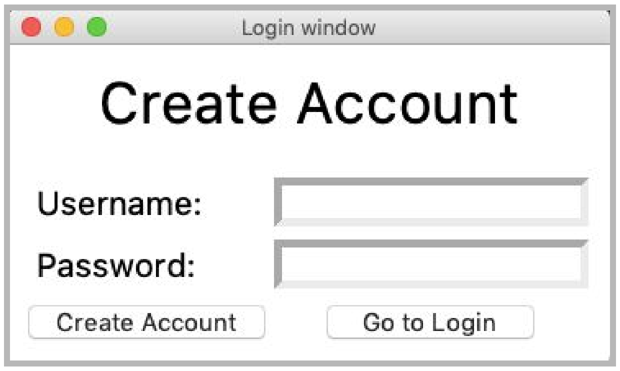
```
# Frame Packing Methods

def log(self):
    self.username.set('')
    self.password.set('')
    self.crf.pack_forget()
    self.head['text'] = 'LOGIN'
    self.logf.pack()

def cr(self):
    self.n_username.set('')
    self.n_password.set('')
    self.logf.pack_forget()
    self.head['text'] = 'Create Account'
    self.crf.pack()
```

In both of log (login) and cr (create) function, the username and password are set to have empty strings, and crf (create function) and logf (login function) have pack_forget which removes the widget from its current manager and can be displayed again by pack or other manager. It is efficient as unnecessary variable can be removed in each function that displays on each window.

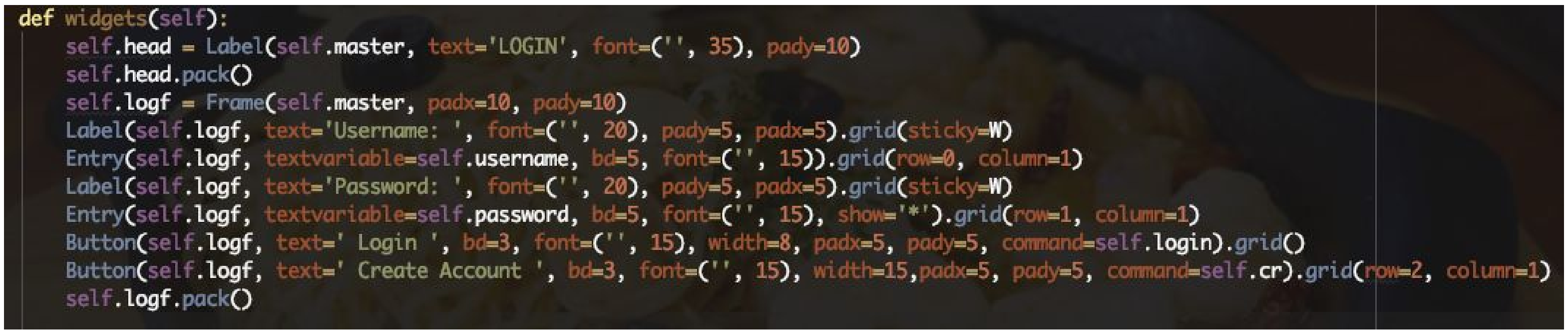
In the LOGIN and Create Account window, there is a head label of LOGIN and Create Account. Both have pack geometry manager for display.

Login & Create Account windows:

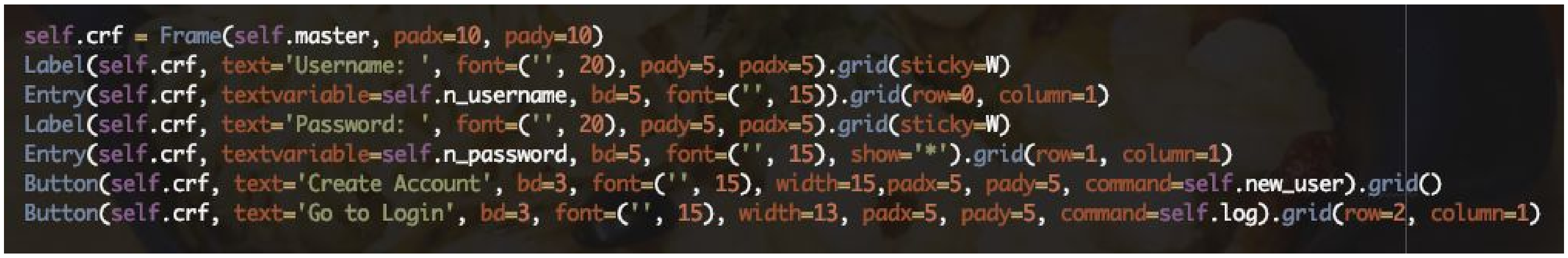




Use of controls:

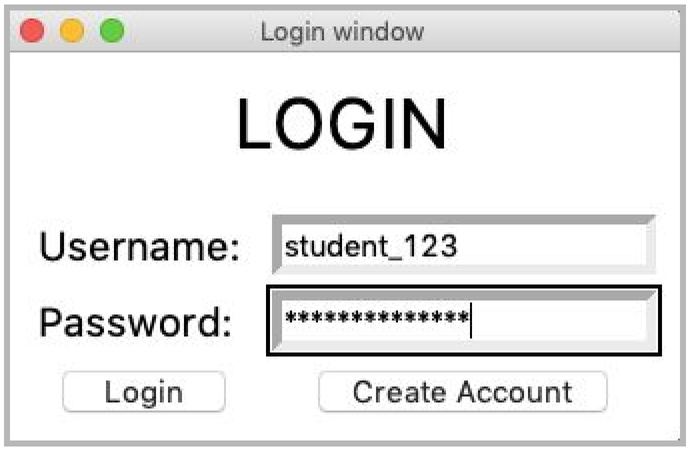


```
def widgets(self):
    self.head = Label(self.master, text='LOGIN', font=('', 35), pady=10)
    self.head.pack()
    self.logf = Frame(self.master, padx=10, pady=10)
    Label(self.logf, text='Username: ', font=('', 20), pady=5, padx=5).grid(sticky=W)
    Entry(self.logf, textvariable=self.username, bd=5, font=('', 15)).grid(row=0, column=1)
    Label(self.logf, text='Password: ', font=('', 20), pady=5, padx=5).grid(sticky=W)
    Entry(self.logf, textvariable=self.password, bd=5, font=('', 15), show='*').grid(row=1, column=1)
    Button(self.logf, text=' Login ', bd=3, font=('', 15), width=8, padx=5, pady=5, command=self.login).grid()
    Button(self.logf, text=' Create Account ', bd=3, font=('', 15), width=15,padx=5, pady=5, command=self.cr).grid(row=2, column=1)
    self.logf.pack()
```



```
self.crf = Frame(self.master, padx=10, pady=10)
Label(self.crf, text='Username: ', font=('', 20), pady=5, padx=5).grid(sticky=W)
Entry(self.crf, textvariable=self.n_username, bd=5, font=('', 15)).grid(row=0, column=1)
Label(self.crf, text='Password: ', font=('', 20), pady=5, padx=5).grid(sticky=W)
Entry(self.crf, textvariable=self.n_password, bd=5, font=('', 15), show='*').grid(row=1, column=1)
Button(self.crf, text='Create Account', bd=3, font=('', 15), width=15,padx=5, pady=5, command=self.new_user).grid()
Button(self.crf, text='Go to Login', bd=3, font=('', 15), width=13, padx=5, pady=5, command=self.log).grid(row=2, column=1)
```

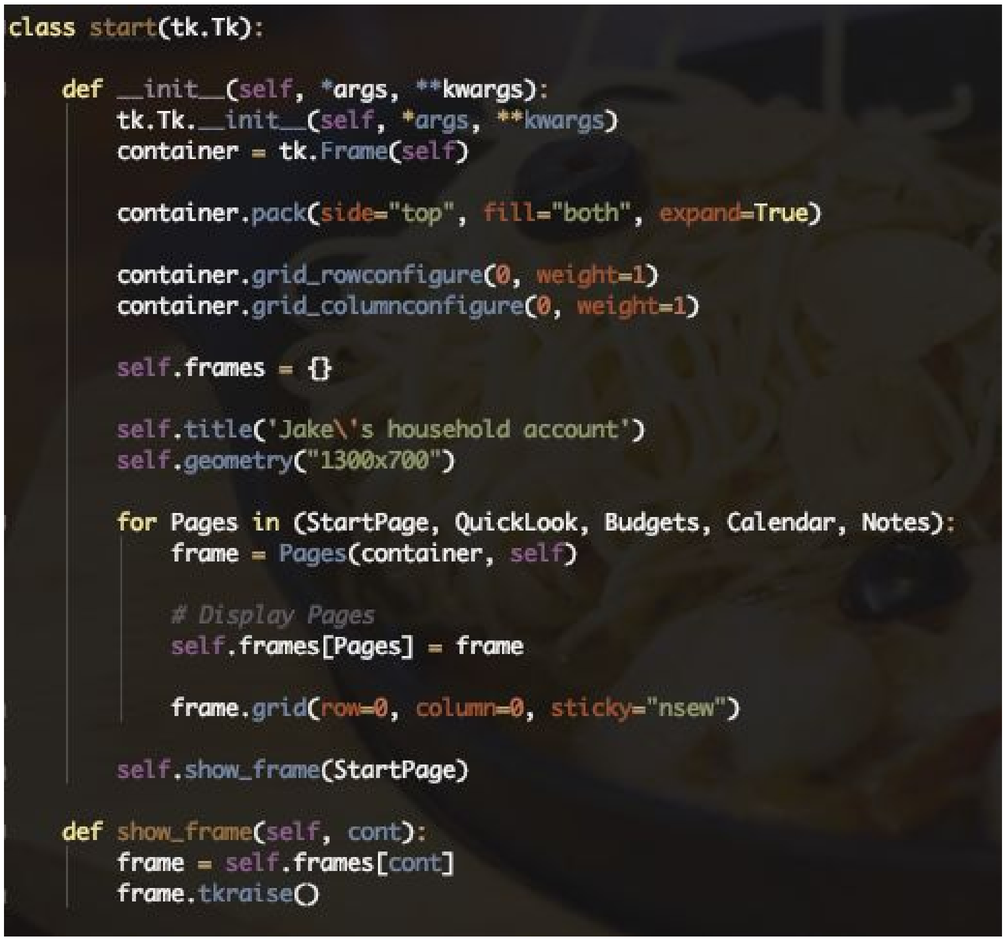
1st and 2nd screenshots are for LOGIN window and Create Account window. Each window is tied to a frame and has labels, entries and buttons to direct to correct windows. Passwords are covered with asterisks.



Username: student_123

Password: **************

- **Household account file -**

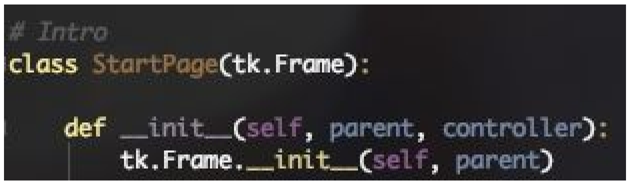


```
class start(tk.Tk):

    def __init__(self, *args, **kwargs):
        tk.Tk.__init__(self, *args, **kwargs)
        container = tk.Frame(self)

        container.pack(side="top", fill="both", expand=True)

        container.grid_rowconfigure(0, weight=1)
        container.grid_columnconfigure(0, weight=1)

        self.frames = {}

        self.title('Jake\'s household account')
        self.geometry("1300x700")

        for Pages in (StartPage, QuickLook, Budgets, Calendar, Notes):
            frame = Pages(container, self)

            # Display Pages
            self.frames[Pages] = frame

            frame.grid(row=0, column=0, sticky="nsew")

        self.show_frame(StartPage)

    def show_frame(self, cont):
        frame = self.frames[cont]
        frame.tkraise()
```

As referenced above, a start class is initialized with `tk.Tk`. `*args, **kwargs` are able to accept multiple arguments and key arguments. Since it was imported as tk, it needs to have tk for everything and `container` is the main Frame that packs everything as `side="top", fill="both"`. `self.frames = {}` is a dictionary form so the windows (pages) can be read in the loop that `frame = Pages(container, self)` and `self.frames[Pages] = frame` to display the windows (pages). [] is a list type.

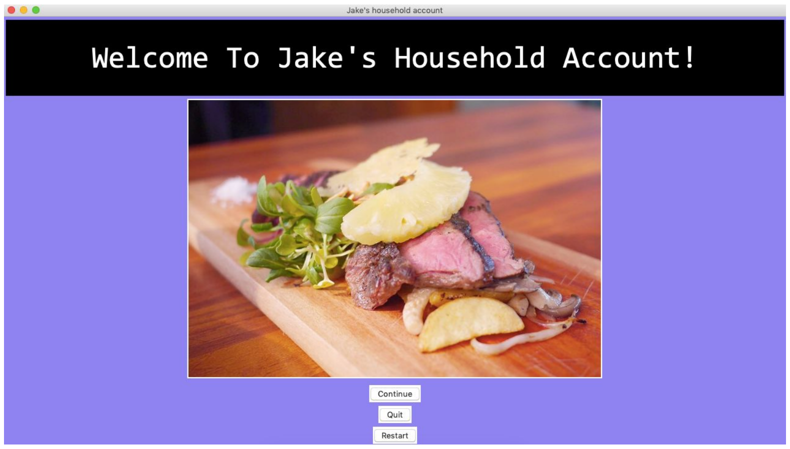
Its first page is set to show StartPage. The function `show_frame` exists to show frames as it continues (`cont`) and `tkraise` means above this.

Class StartPage:

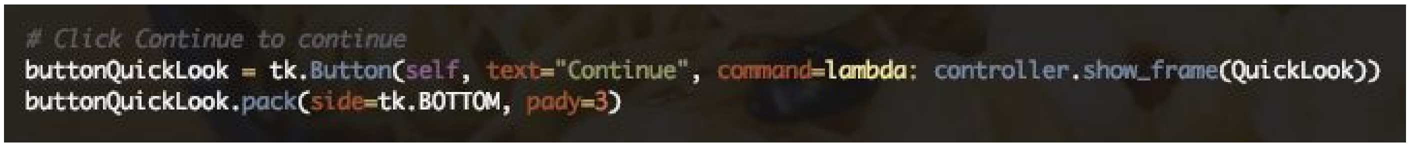


```
# Intro
class StartPage(tk.Frame):

    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
```

The class is initialized and has objects and the class has tk.Frame as a parameter. Since the start class has every frame and tk.Tk as a start, it is possible to have tk.Frame and display it as a different widget.



Continue Button:

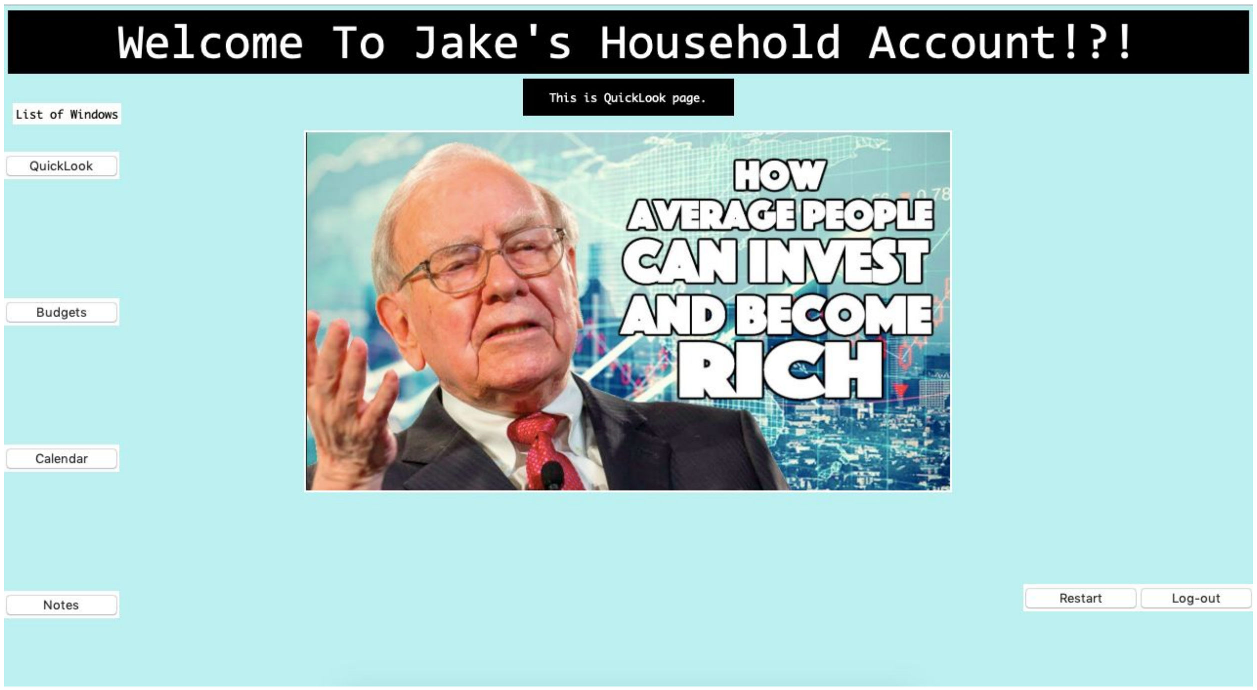


```
# Click Continue to continue
buttonQuickLook = tk.Button(self, text="Continue", command=lambda: controller.show_frame(QuickLook))
buttonQuickLook.pack(side=tk.BOTTOM, pady=3)
```

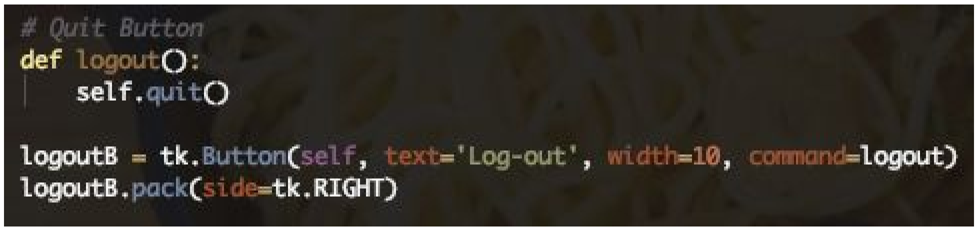
A lambda function is a small anonymous function that can take any number of arguments limited to have only one expression.

controller.show_frame is to direct to the frame of QuickLook.

Page after continue button clicked:


Welcome To Jake's Household Account!?!
This is QuickLook page.
List of Windows
QuickLook
Budgets
Calendar
Notes
HOW AVERAGE PEOPLE CAN INVEST AND BECOME RICH
Restart
Log-out

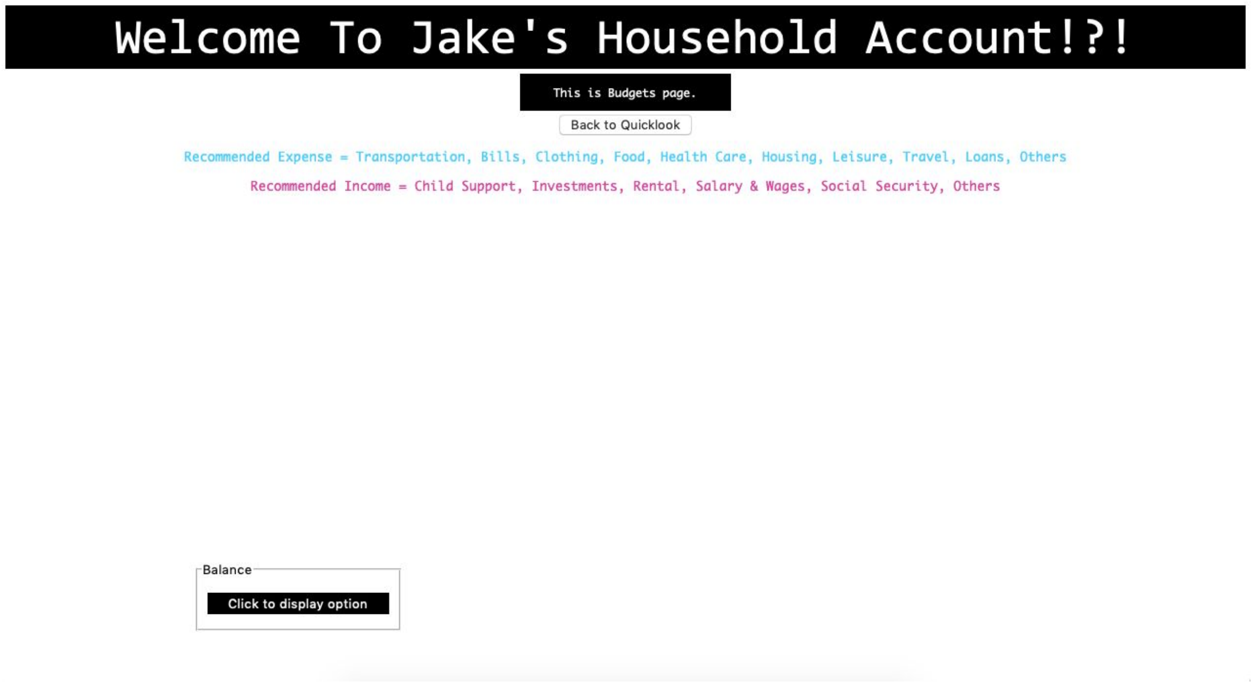
Quit Button:



```
# Quit Button
def logout():
    self.quit()

logoutB = tk.Button(self, text='Log-out', width=10, command=logout)
logoutB.pack(side=tk.RIGHT)
```

logoutB for logging out (quit) and has a command=logout, which directs to a function logout above and executes it, thus quits the program.

Budgets page:



Welcome To Jake's Household Account!?!

This is Budgets page.

Back to Quicklook

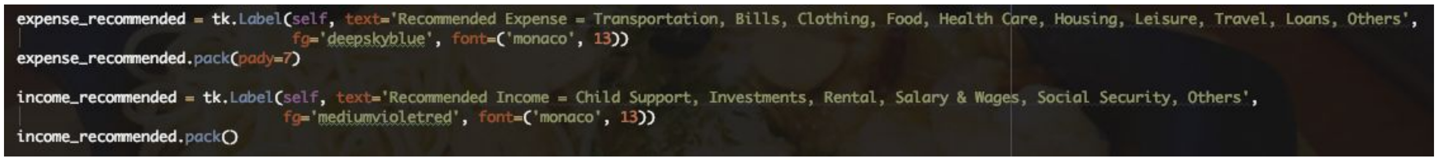
Recommended Expense = Transportation, Bills, Clothing, Food, Health Care, Housing, Leisure, Travel, Loans, Others

Recommended Income = Child Support, Investments, Rental, Salary & Wages, Social Security, Others

Balance

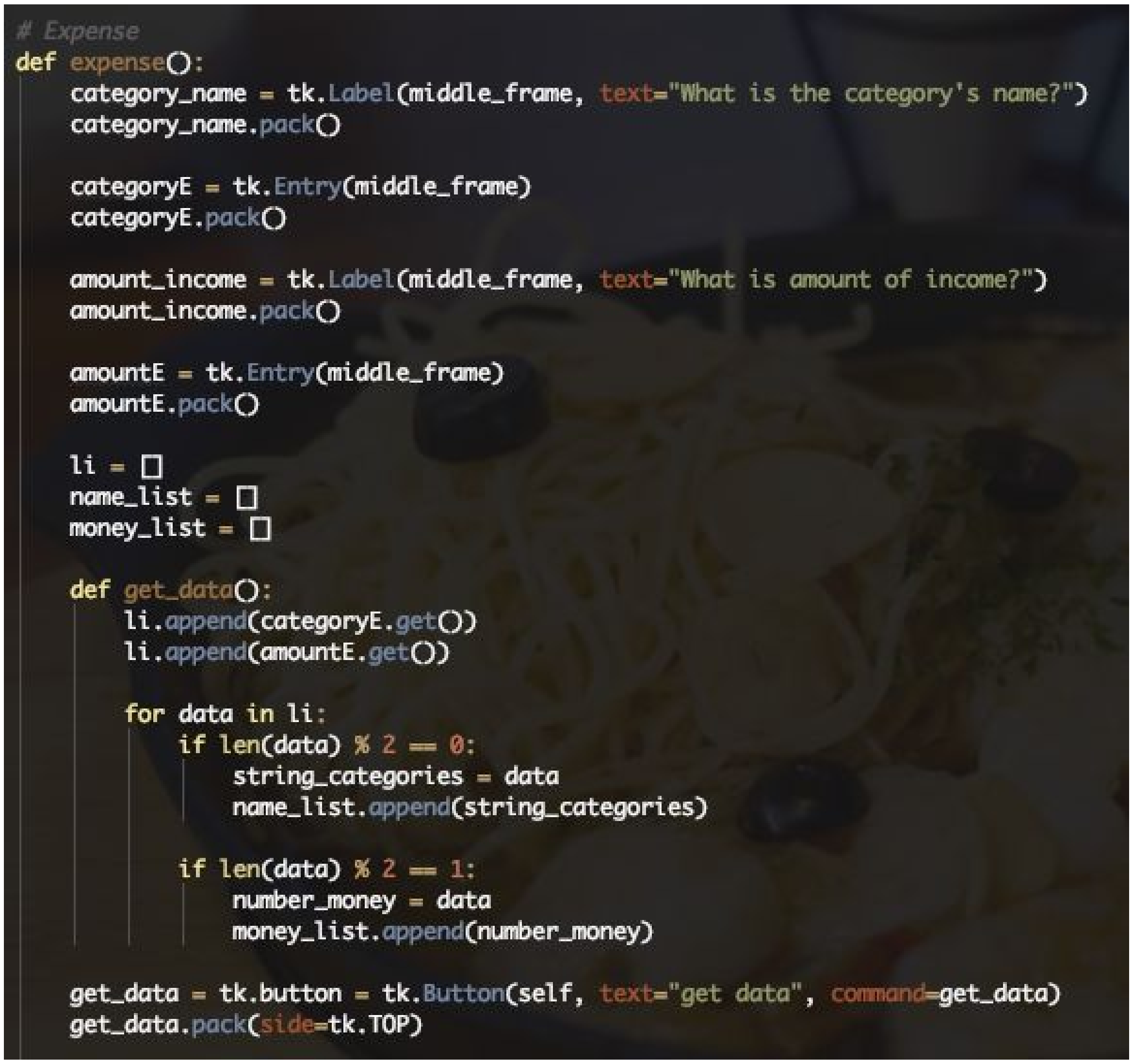
Click to display option

A few codes of expense and income recommendations



```
expense_recommended = tk.Label(self, text='Recommended Expense = Transportation, Bills, Clothing, Food, Health Care, Housing, Leisure, Travel, Loans, Others',
                               fg='deepskyblue', font=('monaco', 13))
expense_recommended.pack(pady=7)

income_recommended = tk.Label(self, text='Recommended Income = Child Support, Investments, Rental, Salary & Wages, Social Security, Others',
                              fg='mediumvioletred', font=('monaco', 13))
income_recommended.pack()
```

Getting input data:



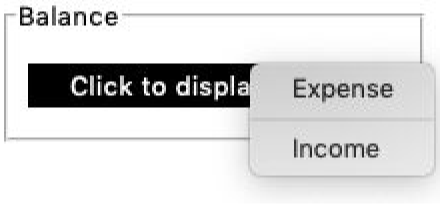
```
# Expense
def expense():
    category_name = tk.Label(middle_frame, text="What is the category's name?")
    category_name.pack()

    categoryE = tk.Entry(middle_frame)
    categoryE.pack()

    amount_income = tk.Label(middle_frame, text="What is amount of income?")
    amount_income.pack()

    amountE = tk.Entry(middle_frame)
    amountE.pack()

    li = []
    name_list = []
    money_list = []

    def get_data():
        li.append(categoryE.get())
        li.append(amountE.get())

        for data in li:
            if len(data) % 2 == 0:
                string_categories = data
                name_list.append(string_categories)

            if len(data) % 2 == 1:
                number_money = data
                money_list.append(number_money)

    get_data = tk.button = tk.Button(self, text="get data", command=get_data)
    get_data.pack(side=tk.TOP)
```

To get input data from entries entered, 3 empty lists are defined as li, name_list and money_list. When get_data button is executed, it directs to get_data function and li gets both data of category's name and amount of money.

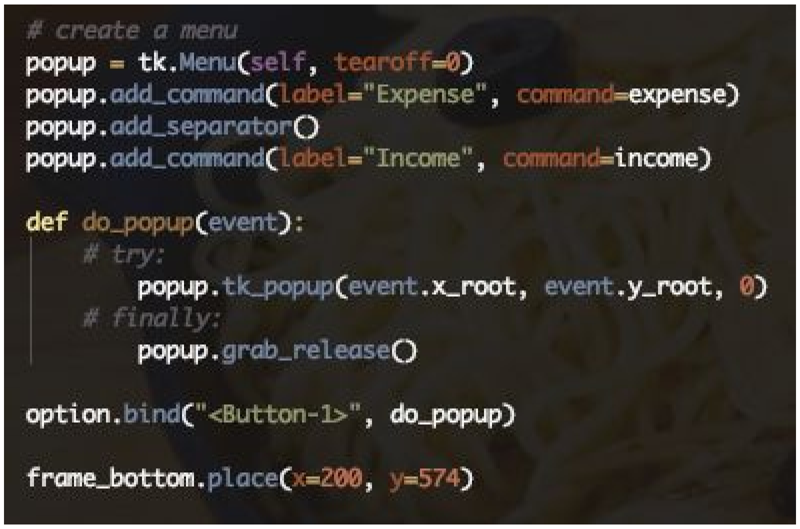
In the loop, every element in the li is data and it is divided by odd (categories' names) and even positions (amount of money). When modulus 2 outputs 0 and 1, it is odd and even. Hence the categories' names are stored in name_list and amount of money are stored in money_list separately.

Although there was an attempt to divide the odds and evens using 1 list, li, it ruined the whole program.

When clicked display option: Expense or Income



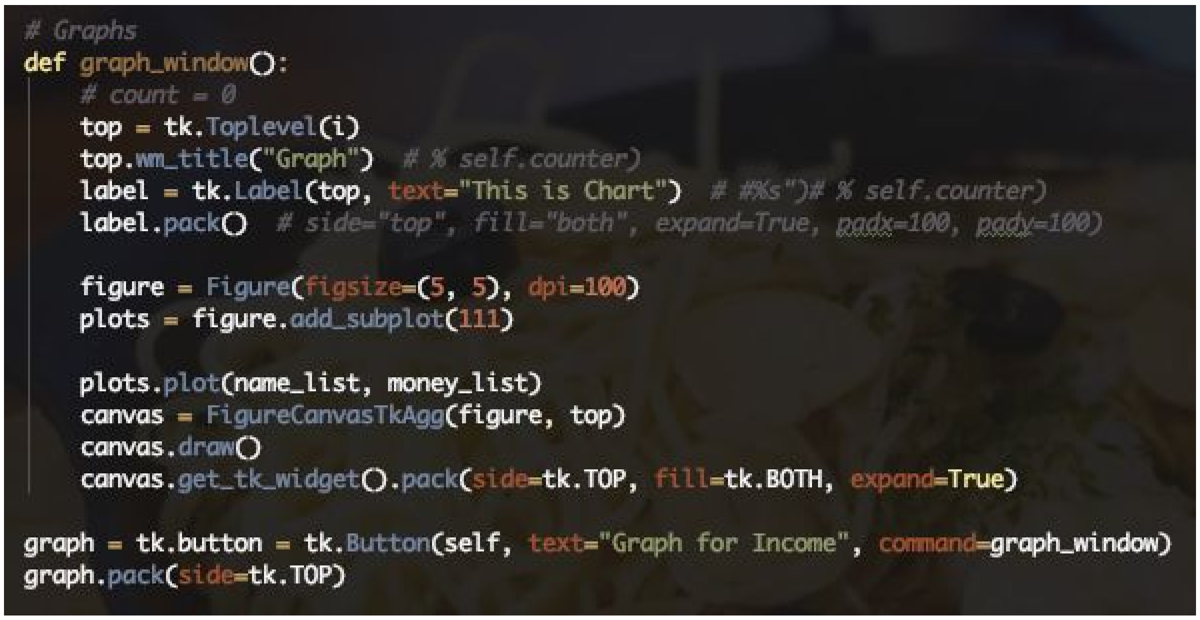
Option Menu:



```
# create a menu
popup = tk.Menu(self, tearoff=0)
popup.add_command(label="Expense", command=expense)
popup.add_separator()
popup.add_command(label="Income", command=income)

def do_popup(event):
    # try:
        popup.tk_popup(event.x_root, event.y_root, 0)
    # finally:
        popup.grab_release()

option.bind("<Button-1>", do_popup)

frame_bottom.place(x=200, y=574)
```

`<Button-1>` is for left-mouse click so if left-mouse click is detected in the label, it shows 2 options of menu, expense and income. The `add_separator` distinguishes them by having a line in-between.

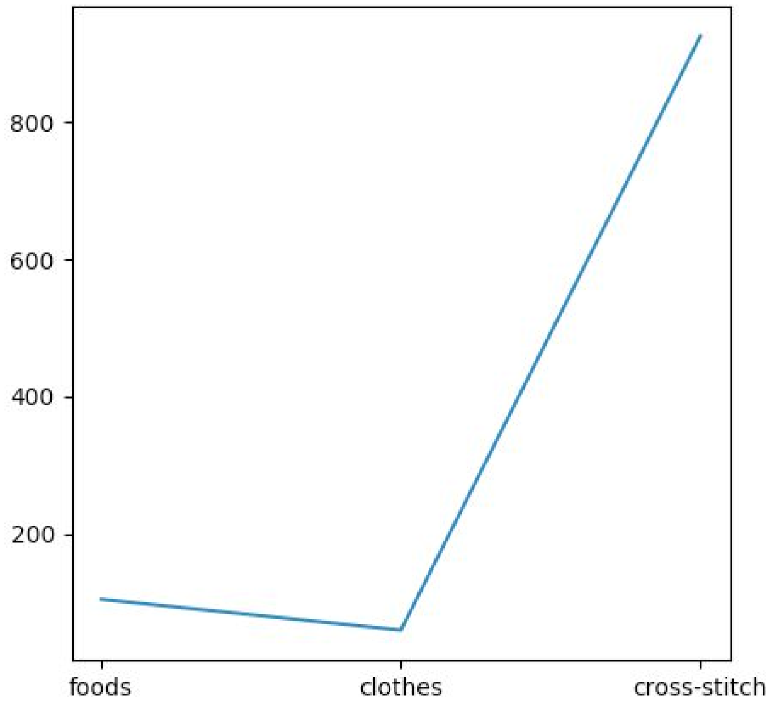
Graph:



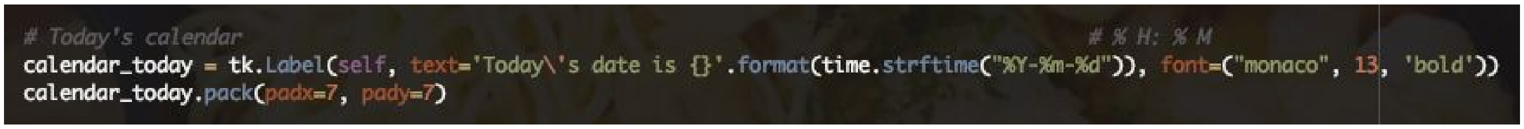
```
# Graphs
def graph_window():
    # count = 0
    top = tk.Toplevel(i)
    top.wm_title("Graph")  # % self.counter)
    label = tk.Label(top, text="This is Chart")  # #%s")# % self.counter)
    label.pack()  # side="top", fill="both", expand=True, padx=100, pady=100)

    figure = Figure(figsize=(5, 5), dpi=100)
    plots = figure.add_subplot(111)

    plots.plot(name_list, money_list)
    canvas = FigureCanvasTkAgg(figure, top)
    canvas.draw()
    canvas.get_tk_widget().pack(side=tk.TOP, fill=tk.BOTH, expand=True)

graph = tk.button = tk.Button(self, text="Graph for Income", command=graph_window)
graph.pack(side=tk.TOP)
```

When it plots data to display a graph, it gets data of name_list on x-axis and money_list on y-axis.

Example

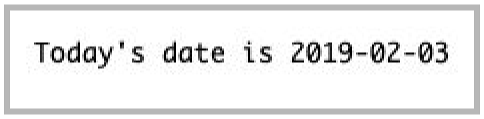


Display today's date:



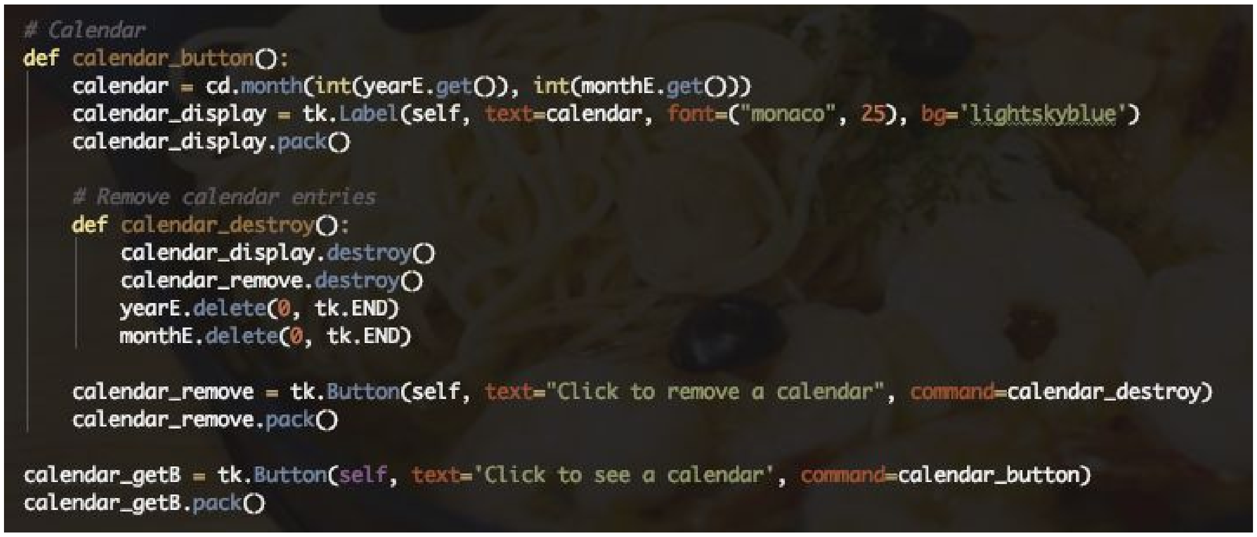
```
# Today's calendar                                                    # % H: % M
calendar_today = tk.Label(self, text='Today\'s date is {}'.format(time.strftime("%Y-%m-%d")), font=("monaco", 13, 'bold'))
calendar_today.pack(padx=7, pady=7)
```

By importing `time` module, it shows today's year, month and date. When `.format()` is used, variables in the string is displayed into `{}` in Today's date is `{}`.



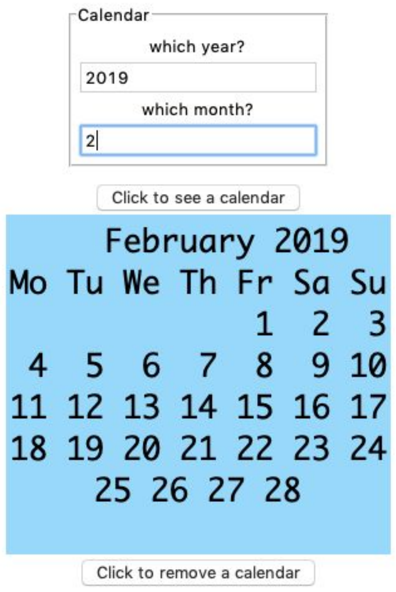
```
Today's date is 2019-02-03
```

Calendar Display:



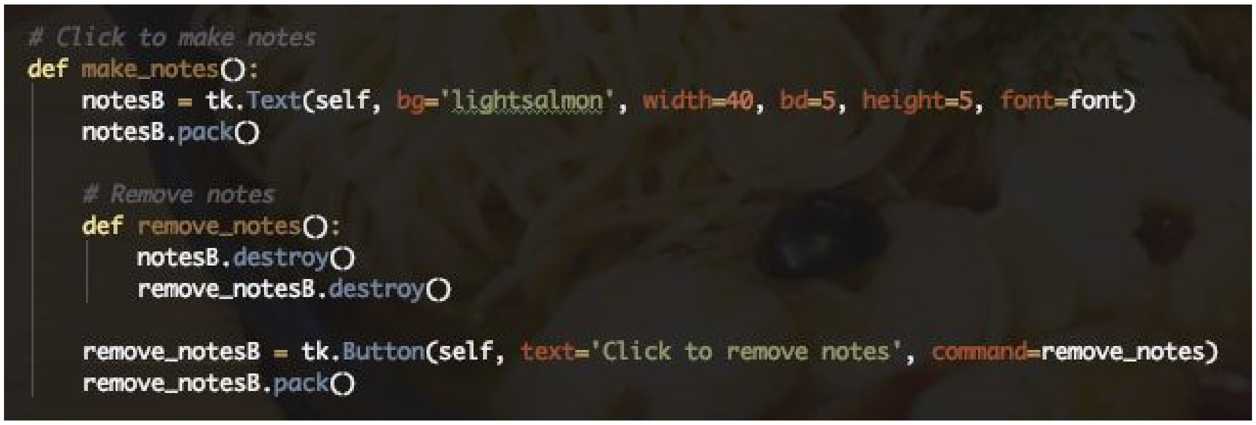
```
# Calendar
def calendar_button():
    calendar = cd.month(int(yearE.get()), int(monthE.get()))
    calendar_display = tk.Label(self, text=calendar, font=("monaco", 25), bg='lightskyblue')
    calendar_display.pack()

    # Remove calendar entries
    def calendar_destroy():
        calendar_display.destroy()
        calendar_remove.destroy()
        yearE.delete(0, tk.END)
        monthE.delete(0, tk.END)

    calendar_remove = tk.Button(self, text="Click to remove a calendar", command=calendar_destroy)
    calendar_remove.pack()

calendar_getB = tk.Button(self, text='Click to see a calendar', command=calendar_button)
calendar_getB.pack()
```

When `calendar_getB` is clicked, it executes function `calendar_button` that the user can input year and month and displays remove button below. When the button `calendar_remove` is clicked, it removes display of the calendar and values entered in the entries.



```
    February 2019
Mo Tu We Th Fr Sa Su
             1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28
```

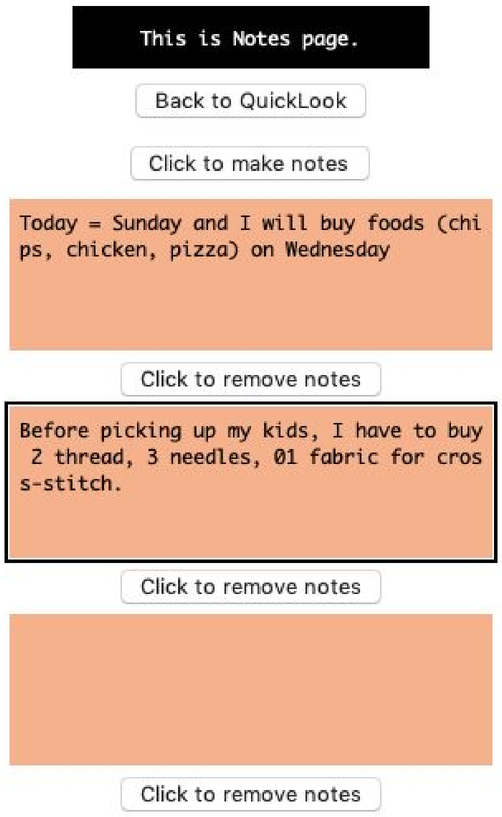
Make Notes:



```
# Click to make notes
def make_notes():
    notesB = tk.Text(self, bg='lightsalmon', width=40, bd=5, height=5, font=font)
    notesB.pack()

    # Remove notes
    def remove_notes():
        notesB.destroy()
        remove_notesB.destroy()

    remove_notesB = tk.Button(self, text='Click to remove notes', command=remove_notes)
    remove_notesB.pack()
```

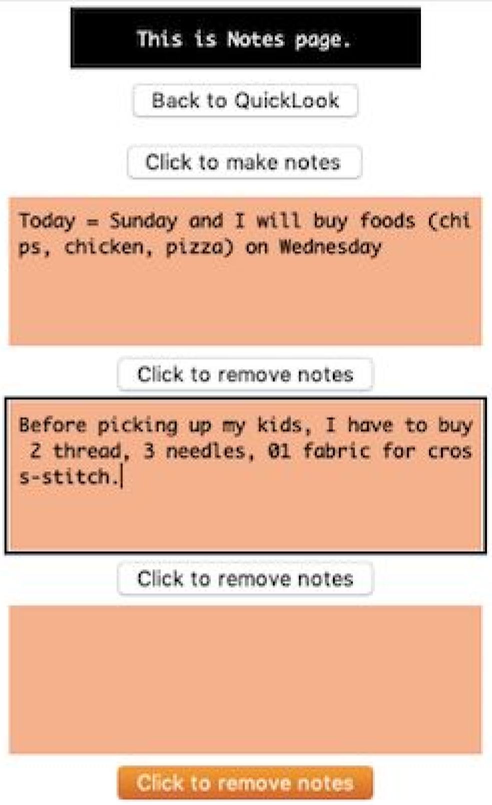
When a note is created, it allows to remove the note above by clicking button `remove_notesB`.

Notes examples


This is Notes page.

Today = Sunday and I will buy foods (chips, chicken, pizza) on Wednesday

Before picking up my kids, I have to buy 2 thread, 3 needles, 01 fabric for cross-stitch.


This is Notes page.

Today = Sunday and I will buy foods (chips, chicken, pizza) on Wednesday

Before picking up my kids, I have to buy 2 thread, 3 needles, 01 fabric for cross-stitch.

Word Count: 1018