**A logo of software department

Description automatically generated**

**College of Engineering, Software Engineering Department**

**SE 1243 Software Development II**

**Laboratories 1 and 2**

**Intro to Objects**

General Instructions: Submit a single python file named “lab1and2.py”. The file should contain the code after doing Parts 3 to 5. You won’t be submitting anything for Parts 1 and 2; they’re just here preparing the way to parts 3 to 5. Submit your .py file to: <https://www.dropbox.com/request/9n7uLvHJj1RdEV37ZVst>

# Part 1:

The previous coding examples for objects have revolved around the Actor object. Using a real-world example makes the concept of classes a little easier to understand. But the Actor class does not accurately reflect how objects are used in programming. The following pages show how objects could be used by a social network like Instagram. Imagine there exists a social network called Photogram that allows you to share, like, and comment on photos. Your feed is comprised of a series of posts containing information like the username, media (image or video), a message, likes, and a list of comments. We are going to create a Post object to reflect this.

Note:

Instagram doesn’t work this way because data for Instagram is stored as JSON, which stands for JavaScript Object Notation. While "object" is a part of JSON, Python objects and JSON objects are different in an important way. JSON objects do not have methods, which will be covered in the next unit. However, using Python objects to represent posts in a social media feed is not far from reality.

It is always a good idea to think about all the various pieces of information that need to be stored in an object. It is also important to think about how that information should be represented. Let’s say you have 100 followers. You could represent that as a string, "100". But storing the follower count as an int is a better idea. If you gain a follower, you cannot say "100" + 1. You would have to typecast "100" as an int, add the new follower, and then typecast the new follower count back to a string. Storing this information as an int is much easier. Here are the elements that make up a post for Photogram:

* Username - The user who creates the post should be stored as a string.
* Id - Some social networks let you change your username. To avoid confusion about usernames, a unique id number is used to refer to each user on a social network. This should be stored as an int.
* Media - Each post has an image or video to display. Media files are often stored elsewhere on a server. The object should store the path to the media file so it can be retrieved and shown to the public. This information should be stored as a string.
* Avatar - The user’s avatar should appear next to their post. The object should store the path to an avatar as a string.
* Comment Button - Each post has a button so viewers can add their comment. The object should store the path to this button as a string. **Note**, this will not be a working button.
* Caption - The caption that accompanies the media file should be stored as a string.
* Likes - The number of times people have liked a post should be stored as an int.
* Comments - Comments should be stored as a string. However, each post can have a multitude of comments. So, this information should be stored as a list of strings.
* Like Button - Heart-shaped icon views could click to like a post. This will be stored as a string.

**Defining the Post Class**

Now that you know all the attributes needed to create a post for Photogram, you can define the Post class.

class Post:

"""Create a post object for the fictitious social network Photogram"""

def \_\_init\_\_(self, username, user\_id, media, avatar, comment\_button, caption, likes, comments, like\_button):

self.username = username

self.user\_id = user\_id

self.media = media

self.avatar = avatar

self.comment\_button = comment\_button

self.caption = caption

self.likes = likes

self.comments = comments

self.like\_button = like\_button

Now, declare an instance of the Post class with some information. For the sake of readability, each of the parameters will be assigned to a variable. Then, the variables will be passed to the object for instantiation. **Note** there is an actual image file that will be used, so be sure the file path is correct.

username = "Sally\_17"

user\_id = 112010

media = "img/photogram/waterfall.png"

avatar = "img/photogram/avatar\_icon.png"

comment\_button = "img/photogram/add\_comment.png"

caption = "First time at Yosemite. It has surpassed all of my expectations."

likes = 23

comments = ["Beautiful!", "I wish I was there too.", "Is that Nevada Falls?", "Love it!", "Can't wait for the Halfdome pictures", "More pics please"]

like\_button = "img/photogram/likes\_icon.png"

post1 = Post(username, user\_id, media, avatar, comment\_button, caption, likes, comments, like\_button)

Print each attribute of post1 to see that everything is working as expected.

# Part 2:

Now that you have all the information needed to make a post, you can turn that into visual output using the Tkinter module. Tkinter allows you to build simple graphical user interfaces with a minimal amount of code. The next part is going to be an introduction to Tkinter and a few of its features.

These labs will only cover a tiny fraction of what can be done with the Tkinter module. The full documentation for Tkinter can be found [here](http://tcl.tk/man/tcl8.5/TkCmd/contents.htm). This documentation is not very user-friendly. A more beginner-friendly way to learn about Tkintercode> is with this YouTube [playlist](https://www.youtube.com/playlist?list=PLCC34OHNcOtoC6GglhF3ncJ5rLwQrLGnV).

**Main Window**

Tkinter's output is a window. There are three steps needed to get a window up and running. First, import the tkinter module. Second, create a window. Third, run the mainloop for the window. The import statement should be at the very top of the program, and the mainloop should come at the very end. The rest of your program will go between these two lines of code.

import tkinter

window = tkinter.Tk()

window.mainloop() #This is the last line of code in your program

That is really all it takes to get a window up and running. You can add a title to the window and set its size with these commands:

import tkinter

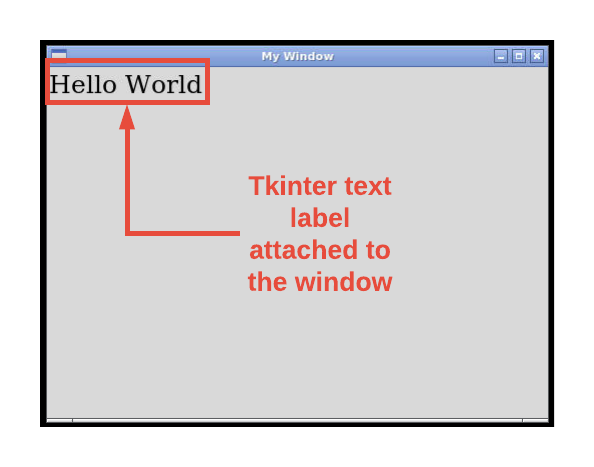
window = tkinter.Tk()

window.title("My Window")

window.geometry("500x350")

window.mainloop() #This is the last line of code in your program

**Label Widget - Text**

Tkinter refers to items added to the main window as widgets. The Label is going to be the building block for these labs.

Create a text label for your window:

my\_label = tkinter.Label(window, text="Hello World", font="DejaVuSerif 18")

Other options for text Labels:

* Font - You can set the font family and size with font options. The available fonts are DejaVuSerif, DejaVuSansMono, DejaVuSans. You can also set the font to bold as well. An example of the font option is: font="DejaVuSeif 18 bold".
* Justify - The justify option allows you to align text inside of a Label. The options are "left", "right", and "center". An example of the justify option is: justify="left".
* Background Color - Use the bg option to set the background color. Colors can be done with either CSS or hex colors. An example of the bg option is: bg="blue".
* Text Color - Use the fg option (foreground) to set the color of the text. An example of the fg option is: fg="red".
* Wrap Length - Widgets set their size based on its contents. While this can be helpful, this can also make getting the perfect layout difficult at times. The wraplength option tells Tkinter when to continue the text on the next line. This keeps the label widget from becoming too wide for your desired layout.

**Grid System**

Adding widgets to the window is a two-step process. First define the widget, then place it in the window using the grid system. The grid system works by positioning widgets with a row and column number. Rows and columns start counting with 0. The sizes of rows and columns depend upon the size of the widget. Here is how to place the my\_label widget in the top-left corner.

my\_label.grid(row=0, column=0)

The grid system can be a bit difficult to use. For instance, if you want to put a single label in location row=1, column=1, the label will appear in the top-left corner. Positioning is relative to other widgets.

**Image Label**

Using an image in a label is also a two-step process. First create an image object for Tkinter, then attach the image to the label by replacing the text option with image. You still need to use grid to place the image in the window.

feather\_image = tkinter.PhotoImage(file="img/feather.png")

image\_label = tkinter.Label(window, image=feather\_image)

image\_label.grid(row=1, column=1)

**Explore**Tkinter

* Try out various fonts and font sizes
* Position labels around the window with the grid system
* Create other image objects (tkinter.PhotoImage) and add more images to the window

# Part 3:

Now that you have had a brief introduction to Tkinter, you are going to create a mockup of a post on Photogram using the information stored in the Post object created in Part 1.

**Setting up the Window**

The first step is to setup the window for the app. Just as in Part 2, you are going to create a window, give it a title, set the size, and choose a background color.

import tkinter

class Post:

"""Create a post object for the fictitious social network Photogram"""

##########

# The rest of the Post class code goes here

##########

window = tkinter.Tk()

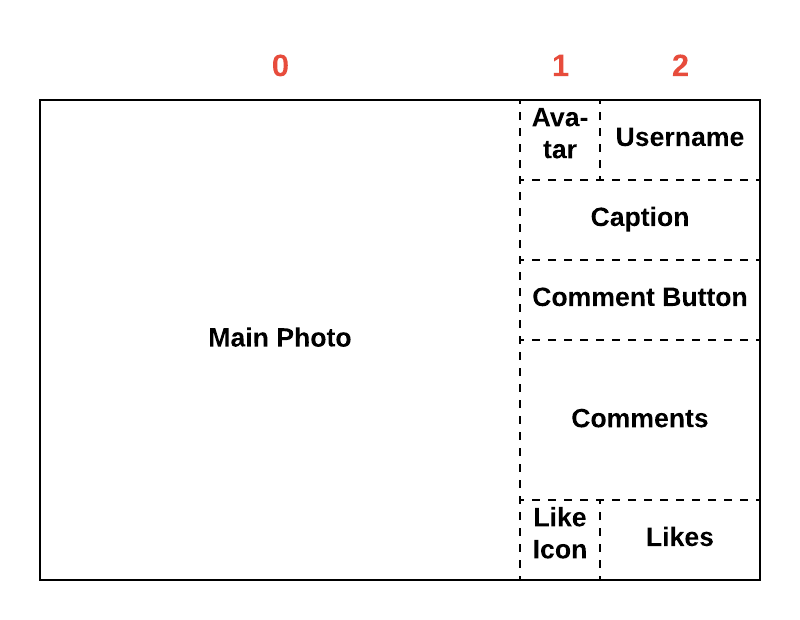
window.title("Photogram")

window.geometry("800x500")

window.configure(background="white")

window.mainloop() # This should be the last line of code in your program

It is important to know in advance what the layout of the app should be. This will influence how the window is created. The image below shows the general layout of Photogram.



It is important to note that the window has three columns. Column 0 will be as wide as the image; Tkinter does this automatically. We want column 2 to be twice as wide as column 1. This will ensure that the avatar and like icon will be positioned properly. Add the following two lines of code to make column 2 twice as wide as column 1. Running the program will show a large, white rectangle. These lines of code should go before window.mainloop().

window.grid\_columnconfigure(1, weight=0)

window.grid\_columnconfigure(2, weight=1)

**Add Photo Variables**

Just like in Part 2, you will need to create a Tkinter image object for each image, and then store them in a variable.

photo = tkinter.PhotoImage(file=post1.media)

comment\_button = tkinter.PhotoImage(file=post1.comment\_button)

avatar = tkinter.PhotoImage(file=post1.avatar)

like\_button = tkinter.PhotoImage(file=post1.like\_button)

**Add Main Photo**

Create the variable image to represent the large image on the left. Adding the image is going to be a two-step process: create the label and place the label.

# Big photo on the left

image = tkinter.Label(

window,

image=photo,

bg="white")

image.grid(

row=0,

column=0,

rowspan=10,

stick="W")

There are a couple of things that were not covered in Part 2. First, rowspan=10 means that the main photo is going to span across ten rows. If you did not do this, row 1 would start below the main photo. In the layout image above, there is nothing below the main photo. stick="W" makes the main photo “sticky” to the “west” (left). If the Tkinter window were to be resized, the main photo would always remain to the far left.

# Part 4:

In this part, you will add the avatar, username, caption, and the comment button to the app. All these labels will follow the same two-step process as before: create the label and then position the label.

**User Avatar**

The avatar should be at the top of the window and just to the right of the main photo. So that would be row 0 and column 1. The avatar image is 30 pixels by 30 pixels, which is why the width is set to 30.

# Gray user avatar

user\_avatar = tkinter.Label(

window,

image=avatar,

width=30,

bg="white")

user\_avatar.grid(

row=0,

column=1,

sticky="W")

**Add Username**

Set the font to DejaVu Sans and make it bold and size 14 for legibility. Making the text left-justified looks better than other positions.

# Username to the right of the avatar

user\_name = tkinter.Label(

window,

text=post1.username,

font="DejaVuSans 14 bold",

justify="left",

bg="white")

user\_name.grid(

row=0,

column=2,

sticky="W")

**Add Caption**

The font in the caption label will be similar to the font for the username, except it will not be bold. The wraplengthparameter determines when the word wrap begins. The bigger the number, the more you can write before word wrap happens. The space to the right of the main photo is actually two columns. We want the caption to fill this entire area, so the text needs to span across two columns.

# User caption for the photo

caption = tkinter.Label(

window,

text=post1.caption,

font="DejaVuSans 14",

wraplength=300,

justify="left",

bg="white")

caption.grid(

row=1,

column=1,

columnspan=2,

sticky="NW")

**Add Comment Button**

The comment button should span across the two columns to the left of the main photo. In addition, the comment button should be centered.

# Add comment icon

comment\_icon = tkinter.Label(

window,

image=comment\_button,

bg="white",

justify="center")

comment\_icon.grid(

row=2,

column=1,

columnspan=2)

# Part 5:

In this part, you will finish up the app by adding comments, an icon for likes, and the number of likes the post has received.

**Add Comments**

The comments are stored in the Python object as a list of strings. The code below will iterate over this list, placing each string into its own label. This label is then attached to the window. Notice that the grid settings are "attached to the Label. That is because no variable name is used for each comment. Previous labels were stored in a variable, like user\_name. So you could say user\_name.Label and list all of the label attributes. Then you could say user\_name.grid and list the grid attributes. Without a variable name, this is not possible. So the label and grid attributes are linked together with a ..

# Loop to add all of the comments

for comment in post1.comments:

tkinter.Label(

window,

text=comment,

font="DejaVuSans 14",

wraplength=300,

justify="left",

bg="white").grid(

row=post1.comments.index(comment) + 3,

column=1,

columnspan=5,

sticky="NW")

**Add Like Icon**

This is the final row in our PhotoGram post. Create an image label for the like button. Place it on row 9 and column 1.

# Add likes icon

likes = tkinter.Label(

window,

image=like\_button,

bg="white",

justify="center")

likes.grid(

row=9,

column=1)

**Add Likes**

The likes count also goes on row 9 and to the right of the like button, so use column 2. The like count should span two columns, but be positioned to the left.

# Add likes count

likes\_count = tkinter.Label(

window,

font="DejaVuSans 14",

text=post1.likes,

bg="white",

justify="left")

likes\_count.grid(

row=9,

column=2,

sticky="W",

columnspan=2)