CS65: Introduction to Computer Science

Graphical User Interface
Errors and Exception
Course Evaluation

Md Alimoor Reza
Assistant Professor of Computer Science

Exercise #1

When the user presses the space bar, create a new ball object, add it to the ball_list, and draw it. The result should be another ball bouncing around the screen

Hint: look at lines 13-15

Exercise #1

When the user presses the space bar, create a new ball object, add it to the ball_list, and draw it. The result should be another ball bouncing around the screen

Hint: look at lines 13-15

Exercise #2

Make the initial location of the ball a random value

Exercise #3

The Ball class has a **`change_color`** method. Implement it so that the ball's color will change when it is called.

Then, add the code in the Driver (Example1.py) so that whenever a ball bounces, it also changes color

Exercise #4

Add the functionality so the ball also increases speed after it bounces. Create a new method called `increase_speed` in the Ball class.

New Material

Graphical User Interfaces

A Graphical User Interface (GUI) is a means for a user to interact with a program which uses graphical components like windows, buttons, and other components that are manipulated with a mouse or touch-input device.

There are several different Python packages/modules that provide the ability to build GUIs (some are designed for desktop/laptop apps, mobile, web, etc.)

`tkinter` is a one of the most popular GUI toolkits (modules), and it works on standard Windows, Mac, and Linux computers.

GUI1.py

Note the use of objects and classes here :)

GUI1.py

```
1  import tkinter
2
3  class MinimalApp:
4
5      def __init__(self):
6
7          #create the main window
8          self.main_window = tkinter.Tk()
9
10         #enter the "main loop" to launch the window and interact with the user
11         tkinter.mainloop()
12
13 my_gui = MinimalApp()
```

tkinter widgets

A “widget” is another name for GUI objects (labels, buttons, sliders, etc.)

- Label: for displaying text to the user
- Button: for triggering code when the user clicks it
- Entry: a box that lets the user type short input

There are two things to do with every widget

- create a new attribute for it in your class
- call the widget's `pack()` method - which figures out where to put it and makes it visible

Creating a label

We pass two arguments when creating the tkinter.Label object

- the main window - the parent widget this widget will
- ext that

```
#create the main window
self.main_window = tkinter.Tk()

#create an attribute for our label
self.hello_label = tkinter.Label(self.main_window, text="Hello world!")

#pack the label
self.hello_label.pack()
```

Creating a label

We pass two arguments when creating the tkinter.Label object

- the main window - the parent widget this widget will

```
#create the main window
self.main_window = tkinter.Tk()

#create an attribute for our label
self.hello_label = tkinter.Label(self.main_window, text="Hello world!")

#pack the label
self.hello_label.pack()
```

  - an optional argument, text="Hello world!" - the text that will appear on the label

GUI2.py

GUI2.py

```
1  import tkinter
2
3  class HelloApp:
4
5      def __init__(self):
6
7          #create the main window
8          self.main_window = tkinter.Tk()
9
10         #create an attribute for our label
11         self.hello_label = tkinter.Label(self.main_window, text="Hello world!")
12
13         #pack the label
14         self.hello_label.pack()
15
16         #enter the "main loop" to launch the window and interact with the user
17         tkinter.mainloop()
18
19 my_gui = HelloApp()
```

A GUI with multiple labels

You can include as many labels as you want (and any number of other widgets)

Make a new variable (attribute) for each label

The labels will appear in the order they're packed

A GUI with multiple labels

You can include as many labels as you want (and any number of other widgets)

Make a new variable (attribute) for each label

The labels will appear in the order they're packed

Here's an example with two labels

```
#create attributes for our label
self.hello_label = tkinter.Label(self.main_window, text="Hello world!")
self.description_label = tkinter.Label(self.main_window, text="This is my GUI program.")

#pack the labels
self.hello_label.pack()
self.description_label.pack()
```

GUIMultiLabel.py

GUIMultiLabel.py

```
1  import tkinter
2
3  class MultiLabelApp:
4
5      def __init__(self):
6
7          #create the main window
8          self.main_window = tkinter.Tk()
9
10         #create attributes for our label
11         self.hello_label = tkinter.Label(self.main_window, text="Hello world!")
12         self.description_label = tkinter.Label(self.main_window, text="This is my GUI program.")
13
14         #pack the labels
15         self.hello_label.pack()
16         self.description_label.pack()
17
18
19         #enter the "main loop" to launch the window and interact with the user
20         tkinter.mainloop()
21
22 my_gui = MultiLabelApp()
```

How to find help…

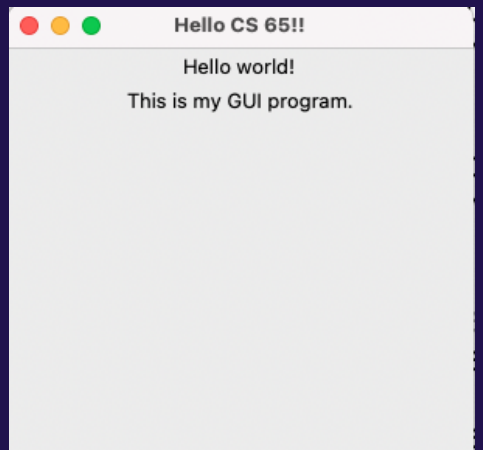
Searching for help on how to do things with tkinter can sometimes be difficult because there are different coding styles people use with it. However, it can be done.

We want to do the following things

- set the title of the window so it is something other than the default tk value
- change the initial size of the window to be something a little bit bigger
- Do some Internet searching (or ChatGPT/DeepSeek) and see if you can figure out how to do this with the GUIMultiLabel.py file given.

Exercise #1

- set the title of the window so it is something other than the default tk value
- change the initial size of the window to be something a little bit bigger
- Do some Internet searching and see if you can figure out how to do this with the `GUIMultiLabel.py` file given.



`tkinter` buttons

Creating a button widget in tkinter is similar to creating a label, except you have to set it up with an extra argument: the name of a function that should run when the button is clicked.

This function is the callback function.

```
self.my_button = tkinter.Button(self.main_window,
                 text="This appears on the button",command=name_of_a_function)
```

Button Demo

```
self.my_button = tkinter.Button(self.main_window,
    text='Click Me!',command=self.do_something)
```

Things to notice about the provided code:

- the command is set to **`self.do_something`**, which is a method we define in the same class
- when we pass the name of the function, we don't include parentheses - it's **not** command=**`self.do_something()`**
- the **`self.do_something()`** function makes a change to the `my_label` attribute using the label's `config` method

```
self.my_button = tkinter.Button(self.main_window,
    text='Click Me!',command=self.do_something)
```

Button Demo

Things to notice about the provided code:

- the command is set to **`self.do_something`**, which is a method we define in the same class
- when we pass the name of the function, we don't include parentheses - it's **not** command=**`self.do_something()`**
- the **`self.do_something()`** function makes a change to the `my_label` attribute using the label's `config` method

```
def do_something(self):
    #the label's config method lets you change the text on the label
    self.my_label.config(text="Good job! You clicked the button.")
```

Show GUIButton1.py

Exercise #2

Take a look at GUIButton2.py

Before running the code, can you predict what will happen?

Exercise #3

When a button is pressed, output a random saying…

Affirmative Answers	Non – Committal Answers	Negative Answers
It is certain	Reply hazy, try again	Don't count on it
It is decidedly so	Ask again later	My reply is no
Without a doubt	Better not tell you now	My sources say no
Yes definitely	Cannot predict now	Outlook not so good
You may rely on it	Concentrate and ask again	Very doubtful
As I see it, yes		
Most likely		
Outlook good		
Yes		
Signs point to yes		

```
class Magic8Ball:
    def __init__(self):
        # Create the main window widget.
        self.main_window = tkinter.Tk()
        self.main_window.title("Magic 8 Ball")
        self.main_window.geometry("400x400")

        # Create a Button widget.
        self.my_button = tkinter.Button(self.main_window,
            text='Click Me!',command=self.do_something)
        self.my_label = tkinter.Label(self.main_window,text="No click yet.")

        # Pack the Button.
        self.my_button.pack()
        self.my_label.pack()

        # Enter the tkinter main loop.
```

```
def do_something(self):
    #the label's config method lets you change the text on the label
    val = random.randint(0,2)
    sayings = ["It is certain!", "Don't count on it.", "Ask again later."]
    self.my_label.config(text=sayings[val])
```

Exceptions

An **exception** is an event that occurs during the execution of a program that disrupts the normal flow of instructions.

Exceptions can be caused by various reasons, such as hardware errors, software bugs (e.g. divide by zero) or user input errors (input a float instead of an int).

What could go wrong?

Exceptions1.py

```
numerator = int(input("Please enter a number: "))
denominator = int(input("Please enter another number: "))
answer = numerator/denominator
print(numerator, "divided by", denominator, "is", answer)
```

Things to know from last class

An **try-except block** is a control structure in programming that is used for exception handling.

The `try` block contains the code that might raise an exception, while the `except` block contains the code that

```
try:
    # code that might raise an exception
except ExceptionType:
    # code to handle the exception
```

Example

```
try:
    numerator = int(input("Please enter a number: "))
    denominator = int(input("Please enter another number: "))
    answer = numerator/denominator
    print(numerator, "divided by", denominator, "is", answer)
except ZeroDivisionError:
    print("denominator cannot be zero")
except ValueError:
    print("please enter an integer")
```

What do you think this code does?

Exceptions.py

```
lock_is_on = True
while lock_is_on == True:
    try:
        number = int(input("please enter an integer: "))
        lock_is_on = False
    except ValueError:
        print("please enter a numeric integer value, e.g. 4")

print("done")
```

In-Class Exercise

Write the code that will prompt the user for an integer, a temperature in Fahrenheit

Use the formula to convert the temperature to Celsius

$$C = (F - 32) * \frac{5}{9}$$

Include try-except blocks to ensure that all possible exceptions are handled.

Copy and paste your code in the In-Class Exercise for May 1.

Course Evaluation

- Finish the survey for this course evaluation
- Link to the evaluation:

https://drake.qualtrics.com/jfe/form/SV_ctCUbWlQizVdqES