UNIT 5 PART -1

EXCEPTION HANDLING

Errors and Exceptions: The programs that we write may behave abnormally or unexpectedly because of some errors and/or exceptions.

Errors:

- The two common types of errors that we very often encounter are *syntax errors* and *logic errors*.

  Syntax errors: And syntax errors, arises due to poor understanding of the language. *Syntax errors* occur when we violate the rules of Python and they are the most common kind of error that we get while learning a new language.

  Example :

```
i=1
while i<=10
    print(i)
    i=i+1
```

  if you run this program we will get syntax error like below,

```
File "1.py", line 2
while i<=10
^
SyntaxError: invalid syntax
```

  Logical errors: While logic errors occur due to poor understanding of problem and its solution. *Logic error* specifies all those type of errors in which the program executes but gives incorrect results. Logical error may occur due to wrong algorithm or logic to solve a particular program.

- However, such errors can be detected at the time of testing.

Exceptions:

• Even if a statement is syntactically correct, it may still cause an error when executed. • Such errors that occur at run-time (or during execution) are known as *exceptions*. • An exception is an event, which occurs during the execution of a program and disrupts the normal flow of the program's instructions.

• Exceptions are run-time anomalies or unusual conditions (such as divide by zero, accessing arrays out of its bounds, running out of memory or disk space, overflow, and underflow) that a program may encounter during execution.

• Like errors, exceptions can also be categorized as synchronous and asynchronous exceptions.

• While synchronous exceptions (like divide by zero, array index out of bound, etc.) can be controlled by the program

• Asynchronous exceptions (like an interrupt from the keyboard, hardware malfunction, or disk failure), on the other hand, are caused by events that are beyond the control of the program.

• When an exception occurs in a program, the program must raise the exception. After that it must handle the exception or the program will be immediately terminated. • if exceptions are not handled by programs, then error messages are generated..

Example:

```
num=int(input("enter numerator"))
den=int(input("enter denominator"))
quo=num/den
print(quo)
```

output:

```
C:\Users\PP>python excep.py
enter numerator1
enter denominator0
```

```
Traceback (most recent call last):
 File "excep.py", line 3, in <module>
 quo=num/den
ZeroDivisionError: division by zero
```

Handling Exceptions:

We can handle exceptions in our program by using try block and except block. A critical operation which can raise exception is placed inside the try block and the code that handles exception is written in except block.

The syntax for try–except block can be given as

```
try:
        statements
except ExceptionName:
        statements
```

Example:

```
num=int(input("Numerator: "))
deno=int(input("Denominator: "))
try:
        quo=num/deno
        print("QUOTIENT: ",quo)
except ZeroDivisionError:
        print("Denominator can't be zero")
```

Output:

```
Numerator: 10
Denominator: 0
```

Denominator can't be zero

Multiple except blocks:

Python allows you to have multiple except blocks for a single try block. The block which matches with the exception generated will get executed.

syntax:

```
try:
    You do your operations here;
    ......................
except(Exception1[, Exception2[,...ExceptionN]]]):
    If there is any exception from the given exception list,
    then execute this block.
    ......................
else:
    If there is no exception then execute this block.
```

e.g.

```
string = input("Enter a String:")
try:
    num = int(input("Enter a number"))
    print(string+num)
except TypeError as e:
    print(e)
except ValueError as e:
    print(e)
```

(OR)

```
string = input("Enter a String:")
```

```
try:
    num = int(input("Enter a number"))
    print(string+num)
except (TypeError,ValueError) as e:
    print(e)
```

OUTPUT :

```
>>>
Enter a String:hai
Enter a number3
Can't convert 'int' object to str implicitly
>>>
Enter a String:hai
Enter a numberbye
invalid literal for int() with base 10: 'bye'
```

Raising Exceptions:

The raise keyword is used to raise an exception.You can define what kind of error to raise, and the text to print to the user**.**

The raise statement allows the programmer to force a specific exception to occur. The sole argument in raise indicates the exception to be raised. This must be either an exception instance or an exception class (a class that derives from Exception)

You can Explicitly raise an exception using the raise keyword.

The general syntax for the raise statement is

raise [Exception [, args [, traceback]]]

Here, Exception is the name of exception to be raised. *args* is optional and specifies a value for the exception argument. If args is not specified, then the exception argument is None. The final argument, *traceback*, is also optional and if present, is the traceback object used for the exception.

```
num=int(input("enter numerator"))
```

```
den=int(input("enter denominator"))
try:
    quo=num/den
    raise Exception("I want an exception anyway")
    print(quo)

except ZeroDivisionError:
    print("Denominator cant be zero")
```

OUTPUT:

```
>>>
enter numerator4
enter denominator0
Denominator cant be zero
>>>
enter numerator4
enter denominator2
Traceback (most recent call last):
  File "C:\Python32\raisex.py", line 5, in <module>
    raise Exception("I want an exception anyway")
Exception: I want an exception anyway
```

Defining clean-up actions(The finally Block)

The finally block is always executed before leaving the try block. This means that the statements written in finally block are executed irrespective of whether an exception has occurred or not.

Syntax:

```
try:
        Write your operations here
        ......................
```

Due to any exception, operations written here will be skipped

finally:

This would always be executed.

.....

e.g.

```
num=int(input("enter numerator"))

den=int(input("enter denominator"))

try:

   quo=num/den

   print(quo)


except ZeroDivisionError:

   print("Denominator cant be zero")

else:

   print("This line is executed when there is no exception")


finally:

    print("TASK DONE")
```

OUTPUT:

```
>>>

enter numerator4

enter denominator2

2.0

This line is executed when there is no exception

TASK DONE
```

>>>

enter numerator4

enter denominator0

Denominator cant be zero

TASK DONE

Built-in and User-defined Exceptions:

Built-in Exceptions:

Exceptions that are already defined in python are called built in or pre-defined exception. In the table listed some built-in exceptions

EXCEPTION NAME	DESCRIPTION
Exception	Base class for all exceptions
ArithmeticError	Base class for all errors that occur for numeric calculation.
OverflowError	Raised when a calculation exceeds maximum limit for a numeric type.
FloatingPointError	Raised when a floating point calculation fails.
ZeroDivisionError	Raised when division or modulo by zero takes place for all numeric types.
EOFError	Raised when there is no input from either the raw_input() or input() function and the end of file is reached.
ImportError	Raised when an import statement fails.
KeyboardInterrupt	Raised when the user interrupts program execution, usually by pressing Ctrl+c.
IndexError KeyError	Raised when an index is not found in a sequence. Raised when the specified key is not found in the dictionary.
NameError	Raised when an identifier is not found in the local or global namespace.

IOError Raised when an input/ output operation fails

SyntaxError	Raised when there is an error in Python syntax.
IndentationError	Raised when indentation is not specified properly.

User –defined exception:

Python allows programmers to create their own exceptions by creating a new exception class. The new exception class is derived from the base class Exception which is predefined in python.

Example:

```
class myerror(Exception):
    def __init__(self,val):
        self.val=val
try:
    raise  myerror(10)
except  myerror  as  e:
    print("user defined exception generated with value",e.val)
```

OUTPUT:

user defined exception generated with value 10

UNIT 5

Graphical User Interface:

Python provides various options for developing graphical user interfaces (GUIs). Most important are listed below.

- Tkinter
- wxPython
- JPython

Tkinter Programming

Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Creating a GUI application using Tkinter is an easy task. All you need to do is perform the following steps –

- Import the *Tkinter* module.
- Create the GUI application main window.
- Add one or more of the above-mentioned widgets to the GUI application.
- Enter the main event loop to take action against each event triggered by the user.

Example:

```
From import Tkinter *
top = Tkinter.Tk()
# Code to add widgets will go here...
top.mainloop()
```

This would create a following window –

Tkinter Widgets:

Tkinter provides various controls, such as buttons, labels and text boxes used in a GUI application. These controls are commonly called widgets.

There are currently 15 types of widgets in Tkinter. We present these widgets as well as a brief description in the following table –

- Button
- Canvas
- Check button
- Entry
- Frame
- Label
- List box
- Menu button
- Menu
- Message
- Radio button
- Scale
- Scrollbar
- Text
- Top level.
- Spin box
- Paned Window
- Label Frame
- Tk Message Box

Standard attributes for widgets

- Dimensions
- Colors
- Fonts
- Relief styles
- Bitmaps
- Cursors

Geometry Management:

All Tkinter widgets have access to specific geometry management methods, Tkinter exposes the following geometry manager classes: pack, grid, and place. •

- The pack() Method - This geometry manager organizes widgets in blocks before placing them in the parent widget.
- The grid() Method - This geometry manager organizes widgets in a table-like

structure in the parent widget.

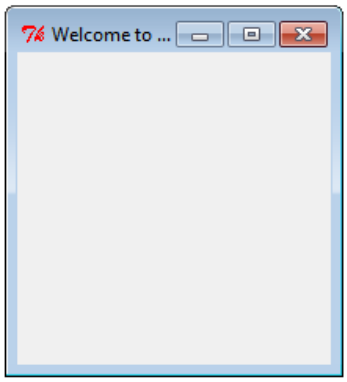
- The place() Method -This geometry manager organizes widgets by placing them in a specific position in the parent widget.

1) Creation of a window/widget

First, we will import Tkinter package and create a window and set its title. The last line which calls mainloop function, this function calls the endless loop of the window, so the window will wait for any user interaction till we close it. If you forget to call the mainloop function, nothing will appear to the user.

Program:

```
from tkinter import *
window = Tk()
window.title("Welcome to tkinter")
window.mainloop()
```



2) Creating a window with specific dimensions and a label

To add a label to our previous example, we will create a label using the label class like this:

```
lbl = Label(window, text="Hello")
```

Then we will set its position on the form using the grid function and give it the location like this:

```
lbl.grid(column=0, row=0)
```

So the complete code will be like this:

Program

```
from tkinter import *
window = Tk()
window.geometry("500x600")
window.title("CSE")
lbl = Label(window, text="HelloWorld",font=("Arial Bold", 50))
lbl.grid(column=0, row=0)
window.mainloop()
```

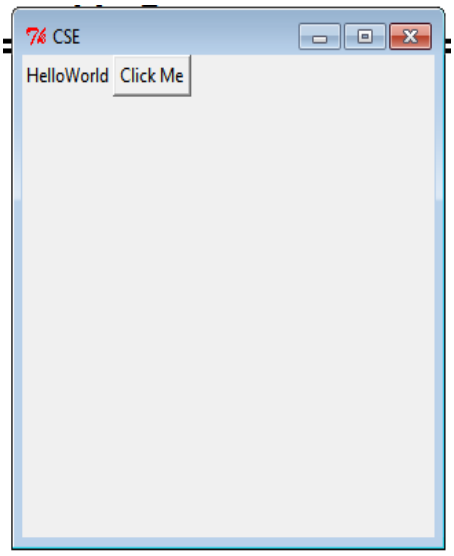
3)Adding a Button to window/widget

Let's start by adding the button to the window, the button is created and added to the window the same as the label:

```
btn = Button(window, text="Click Me")
btn.grid(column=1, row=0)
```

Program:

```
from tkinter import *
window = Tk()
window.geometry("300x300")
window.title("CSE")
lbl = Label(window, text="HelloWorld")
lbl.grid(column=0, row=0)
btn = Button(window, text="Click Me")
btn.grid(column=1, row=0)
window.mainloop()
```



4)Creating 2 text fields to enter 2 numbers, and a button when clicked gives sum of the 2 numbers and displays it in 3rd text field

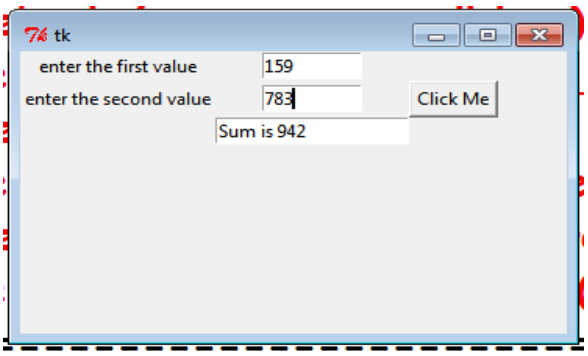
You can create a textbox using Tkinter Entry class like this:

= Entry(window,width=10)

Then you can add it to the window using grid function as usual

Program:

```
from tkinter import *
window = Tk()
window.geometry('350x200')
lbl1 = Label(window, text="enter the first value")
lbl1.grid(column=0, row=0)
lbl2 = Label(window, text="enter the second value")
lbl2.grid(column=0, row=1)
txt1 = Entry(window,width=10)
txt1.grid(column=1, row=0)
txt2 = Entry(window,width=10)
txt2.grid(column=1, row=1)
txt3 = Entry(window,width=20)
txt3.grid(column=1, row=2)
def clicked():
        res=int(txt1.get())+int(txt2.get())
        txt3.insert(0,"Sum is {}".format(res))
btn = Button(window, text="Click Me", command=clicked)
btn.grid(column=2, row=1)
window.mainloop()
```


tk
enter the first value
159
enter the second value
783
Click Me
Sum is 942

5) Creating 2 checkboxes

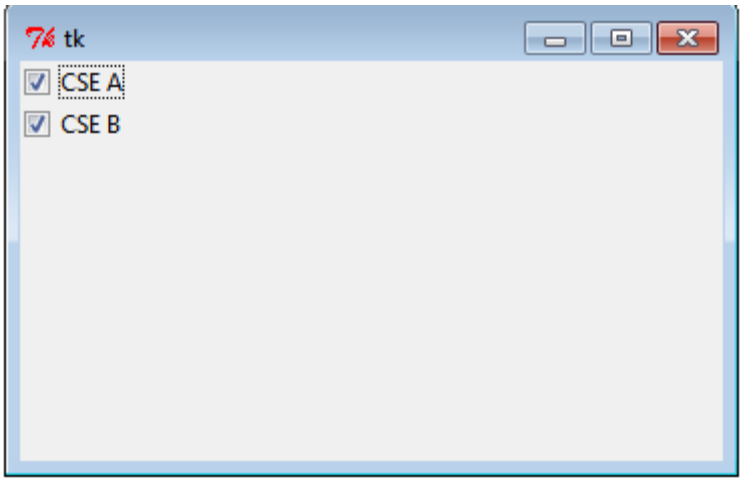
To create a checkbutton, you can use Checkbutton class like this:

```
chk = Checkbutton(window, text='Choose')
```

Here we create a variable of type BooleanVar which is not a standard Python variable, it's a Tkinter variable, and then we pass it to the Checkbutton class to set the check state as the highlighted line in the above example. You can set the Boolean value to false to make it unchecked.

the following program creates a check box.

```
from tkinter import *
from tkinter.ttk import *
window = Tk()
window.geometry('350x200')
chk_state = BooleanVar()
chk_state.set(True) #set check state
chk1 = Checkbutton(window, text='CSE A', var=chk_state)
chk2 = Checkbutton(window, text='CSE B', var=chk_state)
chk1.grid(column=0, row=0)
chk2.grid(column=0, row=1)
window.mainloop()
```



6)Creating 3 radio buttons

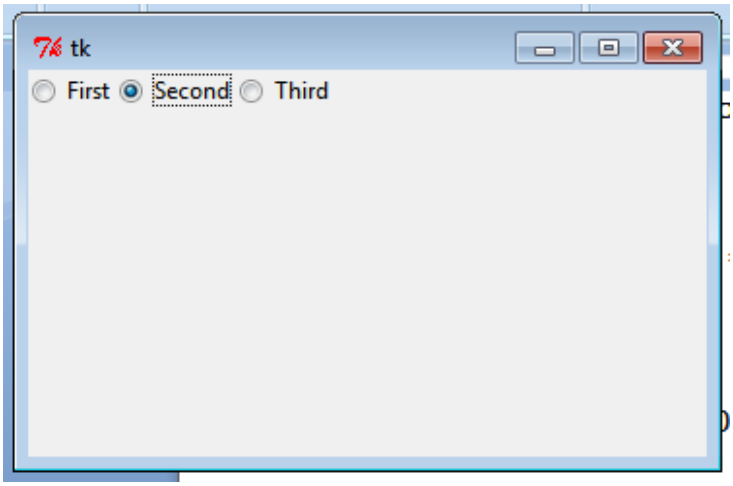
To add radio buttons, simply you can use RadioButton class like this:

rad1 = Radiobutton(window,text='First', value=1)

Note that you should set the value for every radio button with a different value, otherwise, they won't work.

the following program creates a check box

```
from tkinter import *
from tkinter.ttk import *
window = Tk()
window.geometry('350x200')
rad1 = Radiobutton(window,text='First', value=1)
rad2 = Radiobutton(window,text='Second', value=2)
rad3 = Radiobutton(window,text='Third', value=3)
rad1.grid(column=0, row=0)
rad2.grid(column=1, row=0)
rad3.grid(column=2, row=0)
window.mainloop()
```



7) Creating a message box on clicking a button

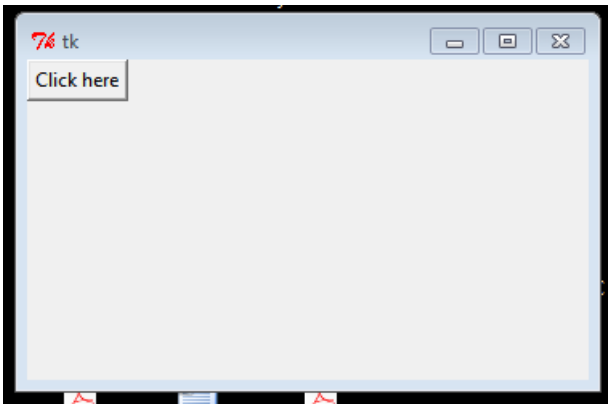
You can show a warning message or error message the same way. The only thing that needs to be changed is the message function.

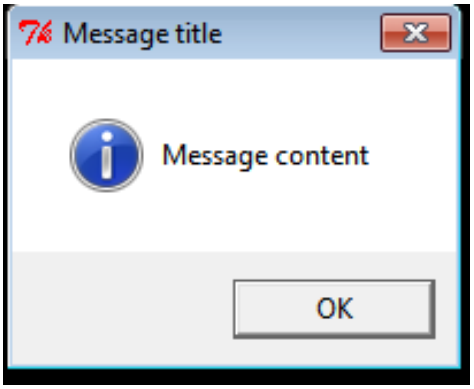
You can show a warning message or error message the same way. The only thing that needs to be changed is the message function

messagebox.showwarning('Message title', 'Message content') #shows warning message

messagebox.showerror('Message title', 'Message content')

```
from tkinter import *
window = Tk()
window.geometry('350x200')
def clicked():
    messagebox.showinfo('Message title ', 'Message content')
    # messagebox.showerror('Message title ', 'Message content')
    # messagebox.show('Message title ', 'Message content')
btn = Button(window,text='Click here', command=clicked)
btn.grid(column=0,row=0)
window.mainloop()
```



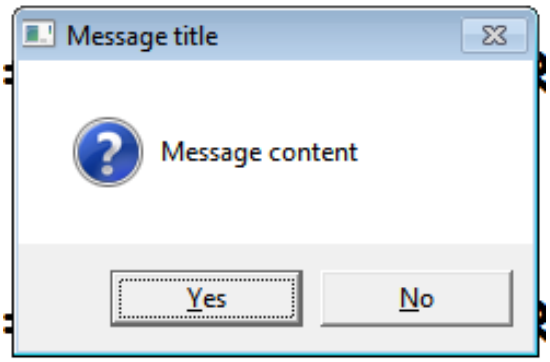

Message title
Message content
OK

8) Creating various message boxes

To show a yes no message box to the user, you can use one of the following messagebox Functions.

- If you click OK or yes or retry, it will return True value, but if you choose no or cancel, it will return False.
- The only function that returns one of three values is askyesnocancel function, it returns True or False or None.

```
from tkinter import messagebox
res = messagebox.askquestion('Message title','Message content')
res = messagebox.askyesno('Message title','Message content')
res = messagebox.askyesnocancel('Message title','Message content')
res = messagebox.askokcancel('Message title','Message content')
res = messagebox.askretrycancel('Message title','Message content')
```



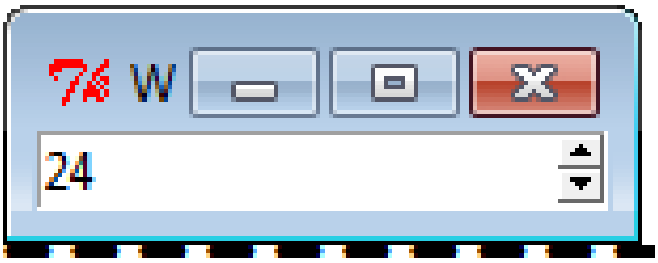
9) Creating a Spinbox

To create a Spinbox widget, you can use Spinbox class like this:

spin = Spinbox(window, from_=0, to=100)

Here we create a Spinbox widget and we pass the from_ and to parameters to specify the numbers range for the Spinbox.

```
from tkinter import *
window = Tk()
window.title("Welcome to tkinter")
spin = Spinbox(window, from_=0, to=100)
spin.grid(column=0,row=0)
window.mainloop()
```



Introduction to programming concepts of scratch

Programming is core of computer science, it's worth taking some time to really get to grips with programming concepts and one of the main tools used in schools to teach these concepts, Scratch.

Programming simply refers to the art of writing instructions (algorithms) to tell a computer what to do. Scratch is a visual programming language that provides an ideal learning environment for doing this. Originally developed by America's Massachusetts Institute of Technology, Scratch is a simple, visual programming language. Colour coded blocks of code simply snap together. Many media rich programs can be made using Scratch, including games, animations and interactive stories. Scratch is almost certainly the most widely used software for teaching programming to Key Stage 2 and Key Stage 3 (learners from 8 to 14 years).

Scratch is a great tool for developing the programming skills of learners, since it allows all manner of different programs to be built. In order to help develop the knowledge and understanding that go with these skills though, it's important to be familiar with some key programming concepts that underpin the Scratch programming environment and are applicable to any programming language. Using screenshots, we will understand the scratch concepts.

Sprites

The most important thing in any Scratch program are the sprites. Sprites are the graphical objects or characters that perform a function in your program. The default sprite in Scratch is the cat, which can easily be changed. Sprites by themselves won't do anything of course, without coding!

Sequences

In order to make a program in any programming language, you need to think through the sequence of steps.

Iteration (looping)

Iteration simply refers to the repetition of a series of instructions. This is accomplished in Scratch using the repeat, repeat until or forever blocks.

Conditional statements

A conditional statement is a set of rules performed if a certain condition is met. In Scratch, the if and if-else blocks check for a condition.

Variables

A variable stores specific information. The most common variables in computer games for example, are score and timer.

Lists (arrays)

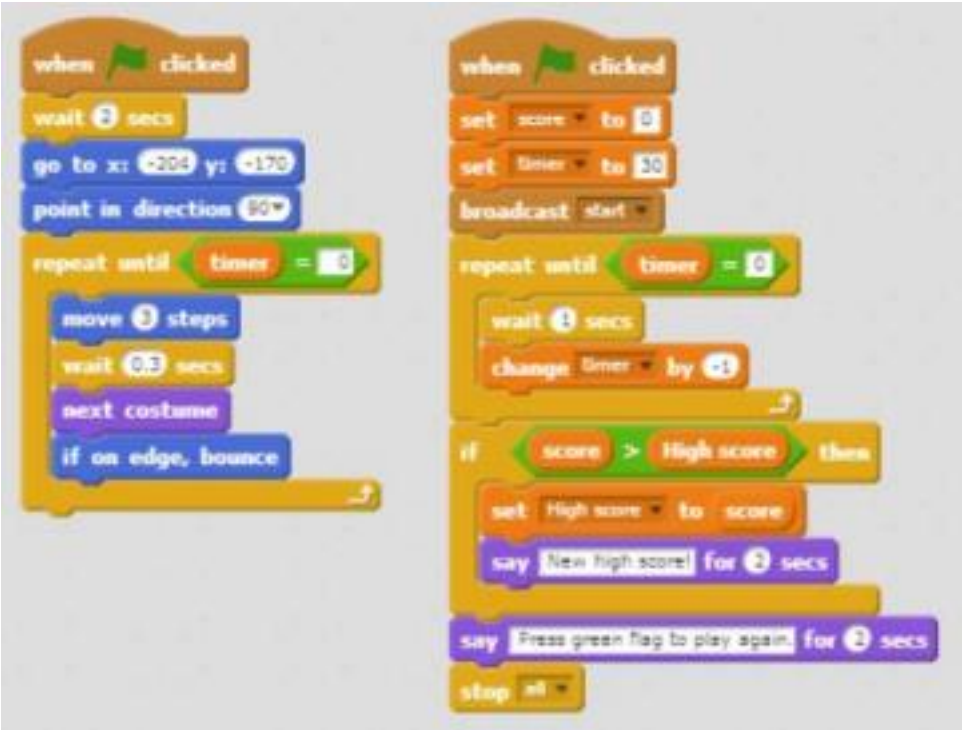
A list is a tool that can be used to store multiple pieces of information at once.

Event Handling

When key pressed and when sprite clicked are examples of event handling. These blocks allow the sprite to respond to events triggered by the user or other parts of the program.

Threads

A thread just refers to the flow of a particular sequence of code within a program. A thread cannot run on its own, but runs within a program. When two threads launch at the same time it is called parallel execution.



Coordination & Synchronisation

The broadcast and when I receive blocks can coordinate the actions of multiple sprites.

They work by getting sprites to cooperate by exchanging messages with one another. A common example is when one sprite touches another sprite, which then broadcasts a new level.

Keyboard input

This is a way of interacting with the user. The ask and wait prompts users to type. The answer block stores the keyboard input.

Boolean logic

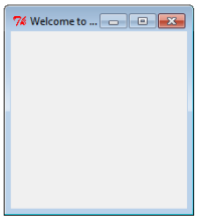
Boolean logic is a form of algebra in which all values are reduced to either true or false. The and, or, not statements are examples of Boolean logic.

User interface design

Interactive user interfaces can be designed in Scratch using clickable sprites to create buttons.

1) Creation of a window/widget

```
from tkinter import *

window = Tk()

window.title("Welcome to tkinter")

window.mainloop()
```

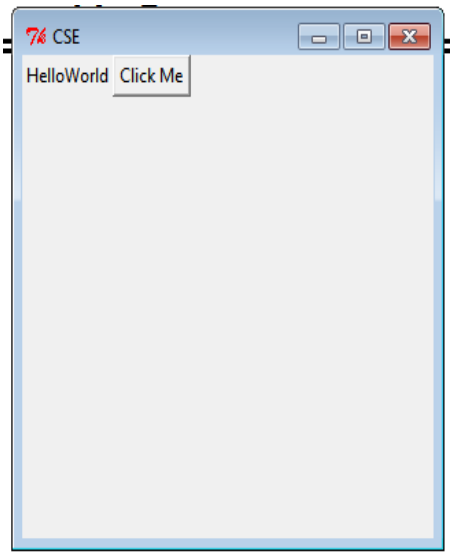


2) Creating a window with specific dimensions and a label

```
from tkinter import *
window = Tk()
window.geometry("500x600")
window.title("CSE")
lbl = Label(window, text="HelloWorld",font=("Arial Bold", 50))
lbl.grid(column=0, row=0)
window.mainloop()
```

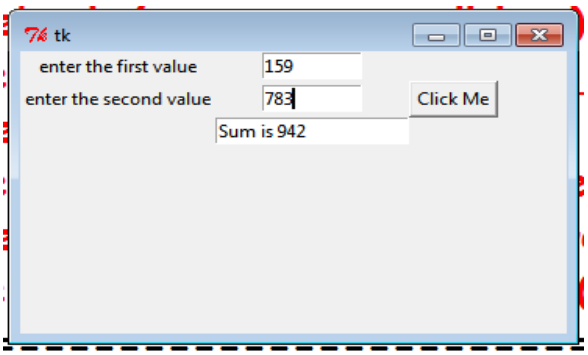
3)Adding a Button to window/widget

```
from tkinter import *

window = Tk()

window.geometry("300x300")

window.title("CSE")

lbl = Label(window, text="HelloWorld")

lbl.grid(column=0, row=0)

btn = Button(window, text="Click Me")
btn.grid(column=1, row=0)

window.mainloop()
```



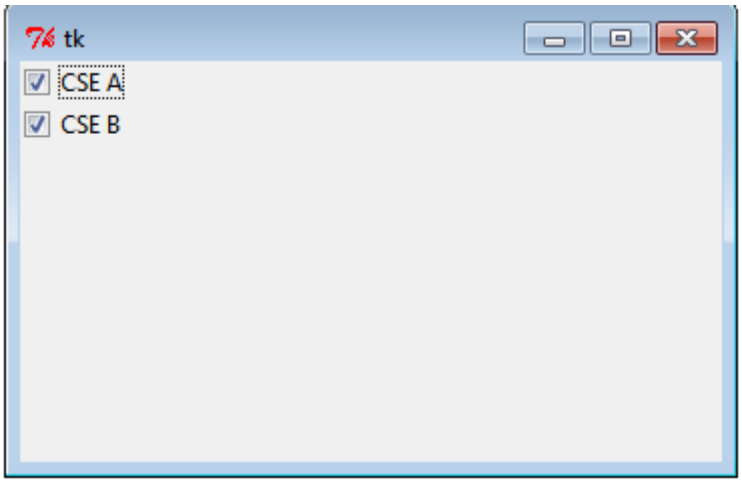
4)Creating 2 text fields to enter 2 numbers, and a button when clicked gives sum of the 2 numbers and displays it in 3rd text field

```
from tkinter import *

window = Tk()

window.geometry('350x200')

lbl1 = Label(window, text="enter the first value")

lbl1.grid(column=0, row=0)

lbl2 = Label(window, text="enter the second value")

lbl2.grid(column=0, row=1)

txt1 = Entry(window,width=10)

txt1.grid(column=1, row=0)

txt2 = Entry(window,width=10)

txt2.grid(column=1, row=1)

txt3 = Entry(window,width=20)

txt3.grid(column=1, row=2)

def clicked():

        res=int(txt1.get())+int(txt2.get())

        txt3.insert(0,"Sum is {}".format(res))

btn = Button(window, text="Click Me", command=clicked)

btn.grid(column=2, row=1)

window.mainloop()
```



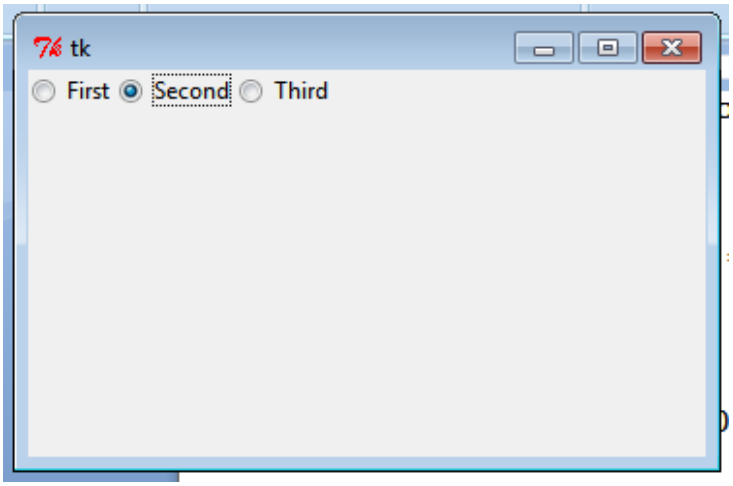
5) Creating 2 checkboxes

```
from tkinter import *
from tkinter.ttk import *
window = Tk()
window.geometry('350x200')
chk_state = BooleanVar()
chk_state.set(True) #set check state
chk1 = Checkbutton(window, text='CSE A', var=chk_state)
chk2 = Checkbutton(window, text='CSE B', var=chk_state)
chk1.grid(column=0, row=0)
chk2.grid(column=0, row=1)
window.mainloop()
```



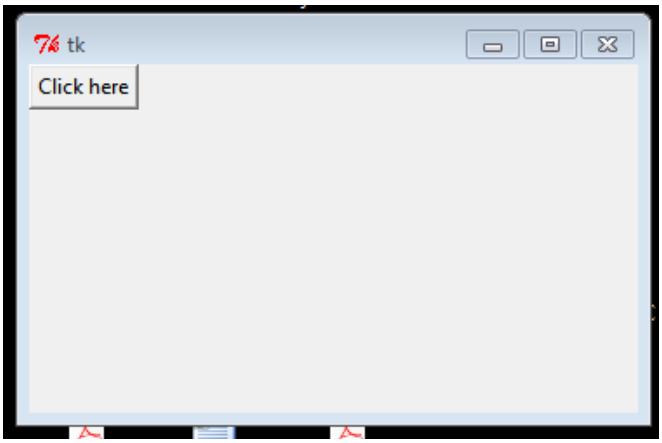
6)Creating 3 radio buttons

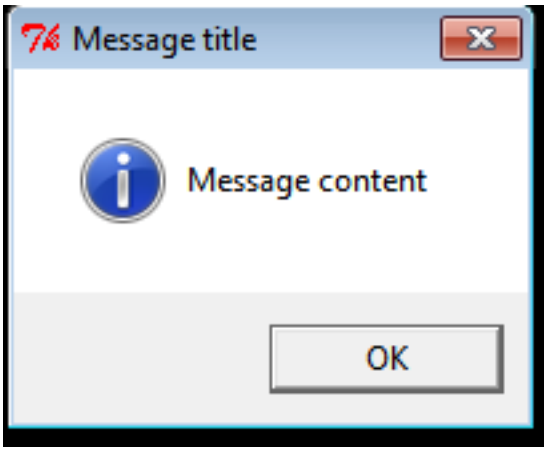
```
from tkinter import *
from tkinter.ttk import *
window = Tk()
window.geometry('350x200')
rad1 = Radiobutton(window,text='First', value=1)
rad2 = Radiobutton(window,text='Second', value=2)
rad3 = Radiobutton(window,text='Third', value=3)
rad1.grid(column=0, row=0)
rad2.grid(column=1, row=0)
rad3.grid(column=2, row=0)
window.mainloop()
```



7) Creating a message box on clicking a button

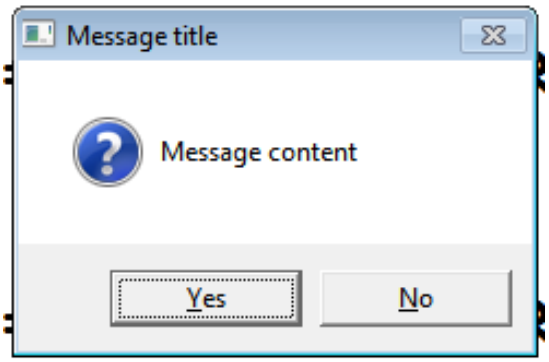
```
from tkinter import *
window = Tk()
window.geometry('350x200')
def clicked():
    messagebox.showinfo('Message title ', 'Message content')
    # messagebox.showerror('Message title ', 'Message content')
    # messagebox.show('Message title ', 'Message content')
btn = Button(window,text='Click here', command=clicked)
btn.grid(column=0,row=0)
window.mainloop()
```





8) Creating various message boxes

```
from tkinter import messagebox

res = messagebox.askquestion('Message title','Message content')

res = messagebox.askyesno('Message title','Message content')

res = messagebox.askyesnocancel('Message title','Message content')

res = messagebox.askokcancel('Message title','Message content')

res = messagebox.askretrycancel('Message title','Message content')
```



9) Creating a Spinbox

```
from tkinter import *
window = Tk()
window.title("Welcome to tkinter")
spin = Spinbox(window, from_=0, to=100)
spin.grid(column=0,row=0)
window.mainloop()
```