At the end of this lecture, students should be able to:

• understand what a GUI is
• start using *tkinter* in Python
• create a Canvas object and draw ovals, rectangles, lines, text, polygons and arcs.
def get_num_uniques(a_sequence):
    uniques = []
    for element in a_sequence:
        if element not in uniques:
            uniques.append(element)
    return len(uniques)

def main():
    desc_1 = "(3, 4, 3, 3, 4, 6, 3, 7, 8, 4):"
    desc_2 = "[3, 4, 3, 3, 4, 6, 3, 7, 8, 4]:"
    print("green apple:", get_num_uniques("green apple"))
    print("abcdefg:", get_num_uniques("abcdefg") )
    print("abbbbbb:", get_num_uniques("abbbbbb") )
    print(desc_1, get_num_uniques((3, 4, 3, 3, 4, 6, 3, 7, 8, 4)))
    print(desc_2, get_num_uniques([3, 4, 3, 3, 4, 6, 3, 7, 8, 4])))

main()
A **Graphical User Interface (GUI)** is a type of interface which allows users to interact with electronic devices through graphical icons and visual indicators, i.e., the visual components of an application or website which aid usability through (hopefully) easily understood graphics and icons.

GUI as opposed to text-based interfaces (which require commands to be typed at the keyboard).

Gui's, gui's everywhere, everywhere gui's
What is a GUI?

- Command line
- Macintosh 128
- Commodore Amiga 500
- Windows 3.1
- Mobile Phone
- Wearable Devices
- Smart Home
- Smart Car
- Smart Refrigerator
- Smart "Things"
- Touchscreens

CompSci 101 - Principles of Programming
Most programs include a GUI and all major programming languages support one or more packages to develop GUIs. 

`tkinter` is not the only GUI-programming toolkit for Python but it is the most commonly used one.
Check that tkinter is installed

Before starting, check that tkinter is properly installed on your system by typing the following in the IDLE interpreter window:

```python
>>> import tkinter
>>> tkinter._test()
```

This command should open a window containing a simple tkinter interface.
A first program using tkinter.

```python
from tkinter import *  # import the tkinter module

def main():
    window = Tk()  # Create an empty window
    window.mainloop()  # Keep the window running
    # until the window is closed

main()
```

This program creates a window which is the top level window to which we will add other components (widgets).

In this program, the variable, `window`, represents the top level window.
What is Tk?

Tk is a robust and platform independent windowing toolkit, and it is available to Python programmers through the tkinter package. Tk provides the definitions of many widgets (labels, buttons, text boxes, menus – the components of a GUI).

In Python, the tkinter package is the interface for Tk. tkinter is a set of wrappers which 'talk' to the Tk widgets and wrap them up as Python objects.

```python
from tkinter import *  #import the tkinter module

def main():
    window = Tk()  #Create an empty window
    window.mainloop()  #Keep the window running
    #until the window is closed
main()
```
A first tkinter program

A background colour for the window can be defined and a title for the window can be defined:

```python
from tkinter import *

def main():
    window = Tk()
    window.title("My first Window")  # Set the title
    window.config(background='blue')  # Set background colour
    window.mainloop()

main()
```

The `config()` method is used to set a variety of different window features, such as the background colour.
The size and position of the window

We would like to control the width, height and position of the top level window. The `geometry()` method is used to set a size for the window and positions it on the screen. The first two arguments are the width and the height in pixels of the window. The last two arguments are x (across) and y (down) screen position coordinates of the window's top left corner.

```python
from tkinter import *

def main():
    window = Tk()
    window.geometry("750x200+10+30")  #width, height, x, y position of the window
    window.title("My first Window")  #Set the title
    window.config(background='purple')  #Set background colour
    window.mainloop()

main()
```
There are lots of 'named colours' which can be used. Look at the website, http://www.science.smith.edu/dftwiki/index.php/Color_Charts_for_TKinter

```python
from tkinter import *

def main():
    window = Tk()
    window.title("My first Window")  #Set the title
    window.config(background='powder blue')  #Set the background colour
    window.geometry("750x200+10+30")
    window.mainloop()

main()
```

Some examples

- light slate gray - gray - light grey - midnight blue
- navy - cornflower blue - dark slate blue - slate blue - medium slate blue - light slate blue - medium blue - royal blue – blue - dodger blue - deep sky blue - sky blue - light sky blue - steel blue - light steel blue - light blue - powder blue
Widgets

GUI interfaces are built by arranging and combining different widgets inside the window.

Widgets are objects which can be added to our top level window. These will allow the user to interact with the program. Some widget examples:

- Buttons, Checkbuttons, Radiobuttons, Menubuttons,
- Entry (for text field entries)
- Message (for displaying text messages to the user)
- **Labels** (text captions, images)
- Frames (a container for other widgets)
- Scale, Scrollbar
- **Canvas** (for drawing shapes, ...)
- Text (for displaying and editing text) and others..

With tkinter we are able to create windows with widgets inside them.

In CompSci 101 we will quickly look at the **Label** widget and then use a **Canvas** widget to draw some shapes.
Adding a Label widget to our top level window

A **Label** widget displays text. Widgets are placed inside parent widgets. In our case the parent of all our widgets is the top level window.

The program below creates three Label widgets, and adds them to the top level window.

```python
from tkinter import *

def main():
    window = Tk()
    window.title("My first Window")
    window.config(background='light sky blue')
    window.geometry("200x150+10+20")
    a_label1 = Label(window, text = "A Label widget in a window")
    a_label2 = Label(window, text = "Another one")
    a_label3 = Label(window, text = "And more!")
    a_label1.pack()
    a_label2.pack()
    a_label3.pack()
    window.mainloop()

main()
```

When creating the Label widget we need to pass the top level widget, `window`, (inside which the label is to be placed) as the first argument.

As well, we need to pass the text which is to be displayed inside the Label.
Arranging the widgets in the window - pack()

Most windowing toolkits (such as tkinter) have layout management systems which works behind the scene and has the job of arranging the widgets in the window!

The `pack()` statement places the Label widget inside the top level window.

```python
from tkinter import *

def main():
    window = Tk()
    window.title("My first Window")
    window.config(background='light sky blue')
    window.geometry("200x150+10+20")
    a_label1 = Label(window, text = "A Label widget in a window")
    a_label2 = Label(window, text = "Another one")
    a_label3 = Label(window, text = "And more!")
    a_label1.pack()
    a_label2.pack()
    a_label3.pack()
    window.mainloop()

main()
```

In Python, layout managers are called Geometry managers and these control the arrangement of the widgets in the window.
Some Label properties - config()

Widgets can be configured (background colour, foreground colour, font) using the config() method.

In tkinter, the **default layout manager** works out the size of each widget and arranges each widget inside their parent (in our case inside the top level window).

```python
from tkinter import *
def main():
    window = Tk()
    window.title("A big Label object")
    window.config(background='light sky blue')
    window.geometry("400x150+10+20")
    a_label = Label(window, text = "A Label widget \nin a window")
    a_label.config(bg="blue", fg="magenta")
    a_font = ("Times", 40, "bold")
    a_label.config(font=a_font)
    a_label.pack()
    window.mainloop()
main()
```
Creating a Canvas widget

A Canvas widget provides a rectangular area inside which shapes (lines, ovals, polygons, rectangles, arcs, text) can be drawn. (The Canvas object can also contain images and bitmaps.)

As well as drawing shapes in the Canvas area, the Canvas object can contain other widgets and frames.

When creating a Canvas widget we need to pass the top level widget, `window`, (inside which the canvas will be placed) as the first argument.

In this program the Geometry manager sets a default size for the Canvas object.

```python
from tkinter import *

def main():
    window = Tk()
    window.title("My first Canvas")
    window.config(background='light sky blue')
    window.geometry("400x300+10+20")

    a_canvas = Canvas(window)
    a_canvas.config(background="green")
    a_canvas.pack()

    window.mainloop()

main()
```
A Canvas widget – expand and fill

Sometimes it is useful to make a widget (such as a Canvas) as big as the parent window and to make the widget resizable when the parent window is resized. The pack() function has optional parameters which control this.

The **expand** optional parameter, if assigned the value **True**, will make the widget expand to the full size of its parent window. (If there is more than one widget in the window, it causes the allocation rectangle to fill the remaining space available in the window).

The **fill** optional parameter, if assigned the value **BOTH**, will make the widget expand to the full size of its allocation rectangle. (This optional parameter can be assigned X to make the widget fill the allocation rectangle horizontally only and it can be assigned Y to make the widget fill the allocation rectangle vertically only.)
A Canvas widget can fill the whole window area

We will make the Canvas object as big as the parent window and make the widget resizable when the parent window is resized.

```python
from tkinter import *

def main():
    window = Tk()
    window.title("My first Canvas")
    window.config(background='light sky blue')
    window.geometry("400x300+10+20")

    a_canvas = Canvas(window)
    a_canvas.config(background="green")  # the Canvas fills the whole top level window
    a_canvas.pack(fill=BOTH, expand = True)

    window.mainloop()

main()
```
from tkinter import *

def draw_in_canvas(a_canvas):
    ... #code which draws inside the Canvas

def main():
    window = Tk()
    window.title("My first Canvas")
    window.config(background='green')
    window.geometry("400x300+10+20")
    a_canvas = Canvas(window)
    a_canvas.config(background="green")
    a_canvas.pack(fill=BOTH, expand = True) #Canvas fills the whole top level window
    draw_in_canvas(a_canvas)
    window.mainloop()
Canvas coordinate system

Each pixel in the Canvas area has an x position (how far from the left of the canvas) and a y position (how far from the top of the canvas). Position (0, 0) is the top left corner of the canvas.
Methods of a Canvas object – draw lines

The Canvas object can be used to draw a line:

```python
create_line(coords, **options)
```

The line is defined by two points, \((x_0, y_0)\), the start position and, \((x_1, y_1)\), the end position.

Note that the line object does not include the end pixel, e.g., the line defined by the coordinates:

\[(0, 10, 5, 10)\]

includes the five pixels:

\[(0, 10), (1, 10), (2, 10), (3, 10), (4, 10)\]

Some line options:

- **fill** (line colour, default is black)
- **width** (line width, default is 1.0)
- **dash** e.g., `dash = (4, 8)` 4 pixels drawn followed by 8 pixels blank
The Canvas object – draw lines

The Canvas object can be used to draw a line:

```python
create_line(coords, **options)
```

Some line options
- **fill** (line colour, default is black)
- **width** (line width, default is 1.0)
- **dash** e.g., dash = (4, 8) 4 pixels drawn followed by 8 pixels blank

```python
from tkinter import *

def lines_in_canvas(a_canvas):
    a_canvas.create_line(0, 0, 100, 200, fill="white")
    a_canvas.create_line(200, 0, 200, 300, fill="purple", width="3.0")
    a_canvas.create_line(30, 0, 130, 200, fill="magenta", width="10.0", dash = (4, 8))

def main():
    window = Tk()
    ...
    lines_in_canvas(a_canvas)
    window.mainloop()

main()
```
The Canvas object – draw rectangles

`create_rectangle(x0, y0, x1, y1, **options)`

The rectangle is defined by two points: (x0, y0) the top left position and (x1, y1) the bottom right position.

Note that the rectangle object:
- does not include the right hand border or the bottom border, e.g., the rectangle with coordinates:
  - (100, 100, 102, 103 )

is 2 pixels by 3 pixels including the 6 pixels:
  - (100, 100), (101, 100),
  - (100, 101), (101, 101),
  - (100, 102), (101, 102),

Some rectangle options:
- `fill` (colour, default is no fill)
- `outline` (The colour of the border)
  Default is `outline='black'`)
- `dash` (dashed border)
  ...
The Canvas object – draw rectangles

create_rectangle(x0, y0, x1, y1, **options)

from tkinter import *

def rectangles_in_canvas(a_canvas):
    a_canvas.create_rectangle(20, 20, 100, 150, fill = "white")
    a_canvas.create_rectangle(200, 150, 250, 250, fill = "blue", dash = (4, 8), outline = "white")
    a_rect = (300, 30, 320, 50)
    a_canvas.create_rectangle(a_rect, fill = "magenta")

def main():
    window = Tk()
    ...
    rectangles_in_canvas(a_canvas)
    window.mainloop()

main()
The Canvas object can be used to draw circles and ovals (called ellipses):

```
create_oval(x0, y0, x1, y1, **options)
```

The oval drawn fits into a rectangle defined by the coordinates:
- (x0, y0) of the top left corner and (x1, y1) of a point just outside of the bottom right corner.

Some oval options
- **fill** (colour, default is no fill)
- **outline** (The colour of the border)
  Default is outline='black’
- **dash** e.g., dash = (4, 8) 4 pixels drawn followed by 8 pixels blank...

[Diagram of oval with coordinates (x0, y0) and (x1, y1)]
The Canvas object – draw ovals

```python
from tkinter import *

def ovals_in_canvas(a_canvas):
    a_canvas.create_oval(20, 20, 100, 150, fill="white", outline="red")
    a_canvas.create_oval(200, 150, 250, 250, fill="blue", dash = (4, 8), outline="white")
    a_box = (300, 30, 320, 50)
    a_canvas.create_oval(a_box, fill="magenta")

def main():
    window = Tk()
    ...
    ovals_in_canvas(a_canvas)
    window.mainloop()
main()
```

Some oval options
- **fill**: (colour, default is no fill)
- **outline**: (The colour of the border. Default is outline='black')
- **dash**: (dashed border)

Some oval options
- **fill**: (colour, default is no fill)
- **outline**: (The colour of the border. Default is outline='black')
- **dash**: (dashed border)
The Canvas object – draw polygons

The Canvas object can be used to draw polygons.

```
cREATE_POLYGON(coords, **options)
```

where the parameter, coords, is a list of points, the x, y values of the polygon.

```
points = [10,10,100,20,70,40,90,50,80,80,180,270]
```
The Canvas object – draw polygons

```
cREATE_POLYGON(coords, **options)
```

from tkinter import *

def polygons_in_canvas(a_canvas):
    points = [10, 10, 100, 20, 70, 40, 90, 50, 80, 80, 180, 270]
    a_canvas.create_polygon(points, fill="white", outline="red")
    coords = [80, 200, 100, 100, 150, 150, 200, 100, 250, 150, 300, 200]
    a_canvas.create_polygon(coords, fill="red", outline="white")

def main():
    window = Tk()
    ...
    polygons_in_canvas(a_canvas)
    window.mainloop()

main()"
The Canvas object – draw text

The Canvas object can be used to display text.

\[
\text{create\_text}(x, y, \text{text}="\ldots", \text{**other\ options})
\]

where the point defined by \(x\) and \(y\) is the position of the text.

The anchor option dictates the position of the text with respect to the point \((x, y)\).

- anchor=NW \((x, y)\) (above and to the right)
- anchor=SW \((x, y)\) (below and going left)
- anchor=NE \((x, y)\) (above and going right)
- anchor=W \((x, y)\) (right)

**Some text options**
- **fill** (colour, default is black)
- **font** (The font used to display the text)
- **anchor** (controls where the text is displayed with respect to \(x, y\))
The Canvas object – draw text

create_text(x, y, text="...", **other options)

from tkinter import *

def text_in_canvas(a_canvas):
    a_canvas.create_line(30, 40, 300, 40)

    a_font = ("Times", 20, "bold")
    a_canvas.create_text(300, 40, text="anchor point is NW", anchor=NW, font=a_font)
    a_canvas.create_text(300, 40, text="anchor point is SE", anchor=SE, font=a_font)

def main():
    window = Tk()
    ...
    text_in_canvas(a_canvas)
    window.mainloop()

main()
The Canvas object – draw arcs

The Canvas object can be used to draw arcs. An arc is a wedge shaped slice of an oval.

```python
create_arc(x0, y0, x1, y1, start=..., extent=..., **other_options)
```

where

- point \((x0, y0)\) is the top left corner and \((x1, y1)\) the lower right corner of a rectangle into which the underlying oval fits.
- **start** is the beginning (degrees) of the arc,
- **extent** is the number of degrees in the arc.

Some other arc options

- **fill** (colour, default is no fill)
- **outline** (The colour of the border. Default is outline='black')
- **dash** (dashed border)

...
from tkinter import *

def arcs_in_canvas(a_canvas):
    start1_degrees = 45
    extent_degrees = 100
    enclosing_oval = (10, 20, 220, 150)
    a_canvas.create_oval(enclosing_oval, fill="white")
    a_canvas.create_arc(enclosing_oval, start=start1_degrees, extent=extent_degrees, fill="red")
    start2_degrees = start1_degrees + extent_degrees + 10
    a_canvas.create_arc(enclosing_oval, start=start2_degrees, extent=extent_degrees, fill="blue")

def main():
    window = Tk()
    ...
    arcs_in_canvas(a_canvas)
    window.mainloop()
main()
In a Python program which uses tkinter, a Canvas object can be used to draw simple shapes

- Line
- Circle
- Rectangle
- Polygon
- Arc
- Text
Examples of Python features used in this lecture

from tkinter import *

def draw_in_canvas(a_canvas):
    points = [10, 10, 100, 20, 70, 40, 90, 50, 80, 80, 180, 270]
    a_canvas.create_polygon(points, fill="white", outline="red")

def main():
    window = Tk()
    window.title("My first Canvas")
    window.config(background='green')
    window.geometry("400x300+10+20")

    a_canvas = Canvas(window)
    a_canvas.config(background="green")
    a_canvas.pack(fill=BOTH, expand = True) #Canvas fills the whole top level window
draw_in_canvas(a_canvas)
    window.mainloop()

main()