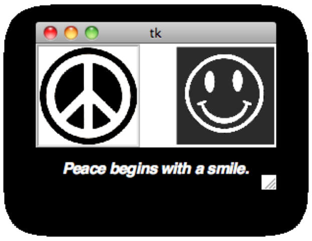
GUI

- Programming Paradigms
- Event-Based `tkinter` Widgets
- Designing GUIs

Packing widgets

Method `pack()` specifies the placement of the widget within its master



Peace begins with a smile.

Option	Description
side	LEFT, RIGHT, TOP, BOTTOM,
fill	'both', 'x', 'y', or 'none'
expand	True or False

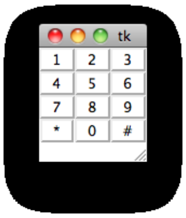
```
from tkinter import Tk, Label, PhotoImage, BOTTOM,
LEFT, RIGHT, RIDGE
root = Tk()

text = Label(root,
             font=('Helvetica', 16, 'bold italic'),
             foreground='white',
             background='black',
             pady=10, padx=25
             text='Peace begins with a smile.')
text.pack(side=BOTTOM)

peace = PhotoImage(file='peace.gif')
peaceLabel = Label(root,
                   borderwidth=3,
                   relief=RIDGE,
                   image=peace)
peaceLabel.pack(side=LEFT)

smiley = PhotoImage(file='smiley.gif')
smileyLabel = Label(root,
                    image=smiley)
smileyLabel.pack(side=RIGHT)

root.mainloop()
```

`smileyPeace.py`

Arranging widgets into a grid

Method `grid()` is used to place widgets in a grid format

`pack()` and `grid()` use different algorithms to place widgets within a master; You must use one or the other for all widgets with the same master.



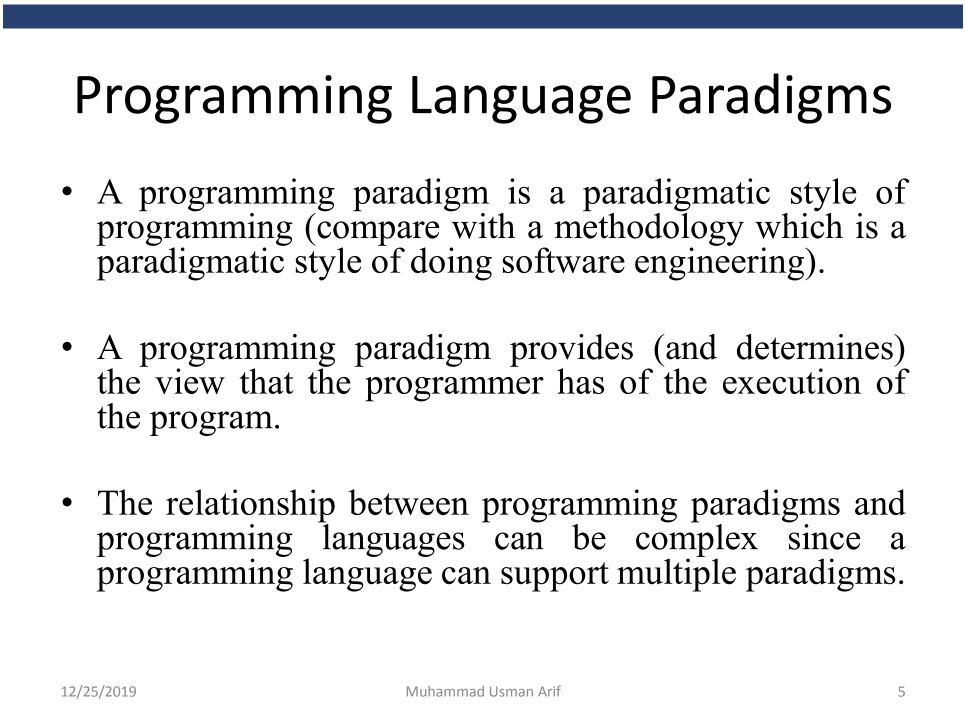
1	2	3
4	5	6
7	8	9
*	0	#

Options
`column`
`columnspan`
`row`
`rowspan`

```
from tkinter import Tk, Label, RAISED

root = Tk()
labels = [['1', '2', '3'],
          ['4', '5', '6'],
          ['7', '8', '9'],
          ['*', '0', '#']]

for r in range(4):
    for c in range(3):
        # create label for row r and column c
        label = Label(root,
                      relief=RAISED,
                      padx=10,
                      text=labels[r][c])
        # place label in row r and column c
        label.grid(row=r, column=c)

root.mainloop()
```

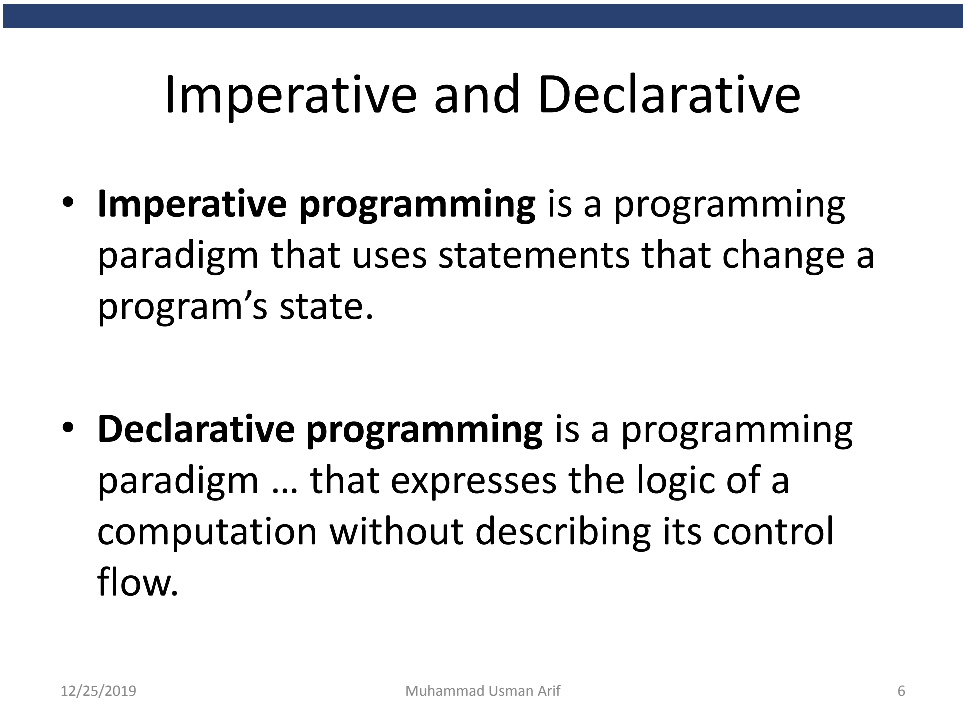
Do you solve problems by just jumping in, willing to ignore the experience and wisdom of those that may have programmed solutions to problems very similar to yours? We learn from the past. Our ancestors discovered and invented ways of programming that we know call paradigms. We benefit from the knowledge they left us, even as we strive to create new paradigms ourselves.

PROGRAMMING PARADIGMS



Programming Language Paradigms

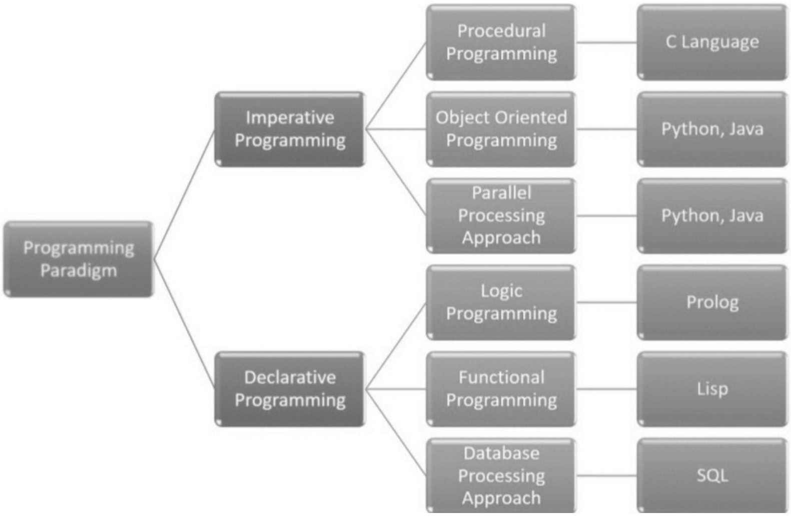
- A programming paradigm is a paradigmatic style of programming (compare with a methodology which is a paradigmatic style of doing software engineering).
- A programming paradigm provides (and determines) the view that the programmer has of the execution of the program.
- The relationship between programming paradigms and programming languages can be complex since a programming language can support multiple paradigms.



Imperative and Declarative

- **Imperative programming** is a programming paradigm that uses statements that change a program’s state.
- **Declarative programming** is a programming paradigm … that expresses the logic of a computation without describing its control flow.

Programming Language Paradigms



Programming Paradigm

Imperative Programming

Procedural Programming — C Language

Object Oriented Programming — Python, Java

Parallel Processing Approach — Python, Java

Declarative Programming

Logic Programming — Prolog

Functional Programming — Lisp

Database Processing Approach — SQL

Imperative

- Procedural
  - Characterized by sequential instructions
  - A program in which statements are grouped into a hierarchy of subprograms
  - Fortran, C, C++
- Object-oriented model
  - Program consists of a set of objects and the interactions among the objects
  - Python, Java, Smalltalk, Simula

Declarative

- Functional
  - Based on the mathematical concept of a function
  - Lisp, Scheme, and ML
- Logic
  - Based on principles of symbolic logic
  - Types of statements
    - declares facts about objects and relationships
    - defines rules about objects
    - asks questions about objects
  - PROLOG

PROLOG

- Pets to owners
  - owns(mary,bo).
  - owns(ann,kitty).
  - owns(bob,riley).
  - owns(susy,charlie).

  - ?-owns(mary,bo)
  - yes
  - ?-owns(bo,mary)
  - no
  - ?-owns(susy,bo)
  - no

States facts

Asks questions

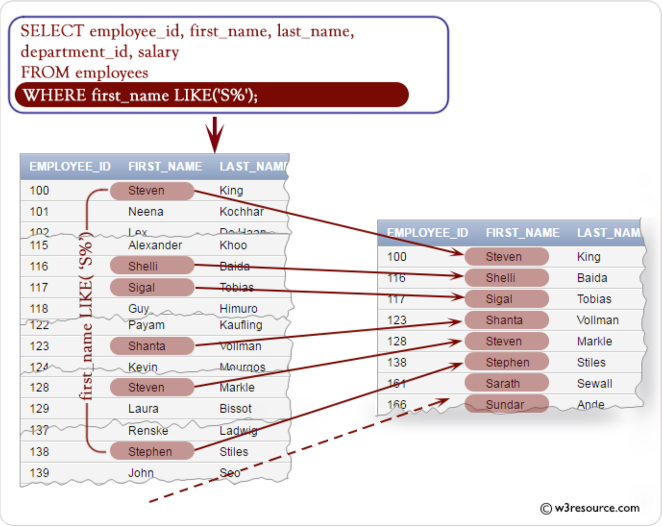
PROLOG

- ?-owns(ann, Cat).
- Cat = kitty

- ?-owns(Name,charlie).
- Name = susy

Upper case is variable; lower case is constant

Database (SQL)



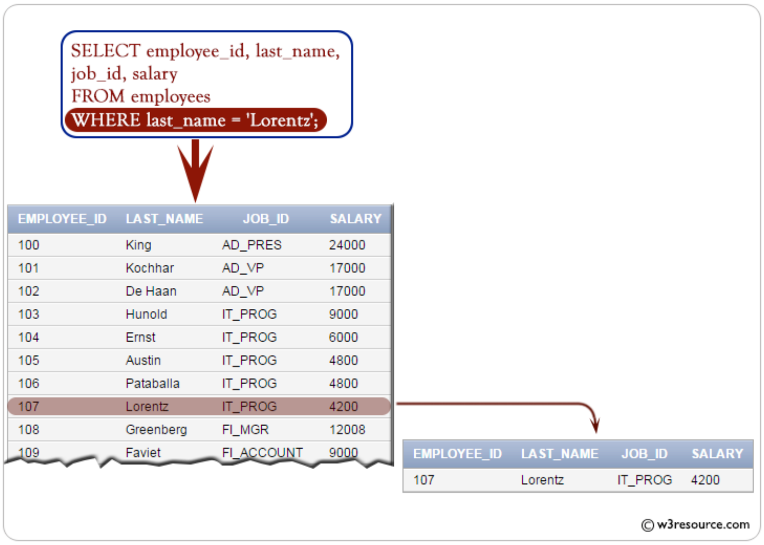
```
SELECT employee_id, first_name, last_name,
department_id, salary
FROM employees
WHERE first_name LIKE('S%');
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME
100	Steven	King
101	Neena	Kochhar
102	Lex	De Haan
115	Alexander	Khoo
116	Shelli	Baida
117	Sigal	Tobias
118	Guy	Himuro
122	Payam	Kaufling
123	Shanta	Vollman
124	Kevin	Mourgos
128	Steven	Markle
129	Laura	Bissot
137	Renske	Ladwig
138	Stephen	Stiles
139	John	Seo

EMPLOYEE_ID	FIRST_NAME	LAST_NAM
100	Steven	King
116	Shelli	Baida
117	Sigal	Tobias
123	Shanta	Vollman
128	Steven	Markle
138	Stephen	Stiles
161	Sarath	Sewall
166	Sundar	Ande

© w3resource.com

Database (SQL)



```
SELECT employee_id, last_name,
job_id, salary
FROM employees
WHERE last_name = 'Lorentz';
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
100	King	AD_PRES	24000
101	Kochhar	AD_VP	17000
102	De Haan	AD_VP	17000
103	Hunold	IT_PROG	9000
104	Ernst	IT_PROG	6000
105	Austin	IT_PROG	4800
106	Pataballa	IT_PROG	4800
107	Lorentz	IT_PROG	4200
108	Greenberg	FI_MGR	12008
109	Faviet	FI_ACCOUNT	9000

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
107	Lorentz	IT_PROG	4200

© w3resource.com

Event-driven programming

When a GUI is started with the `mainloop()` method call, Python starts an infinite loop called an event loop

```
while True:

    1. wait for an event to occur

    2. run the associated event handler
```

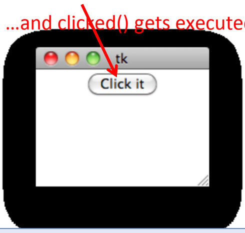
Event-driven programming is the programming approach used to build applications whose execution flow is determined by events and described using an event loop

Widget Button

Widget `Button` represents the standard clickable GUI button

Click the button…

…and clicked() gets executed



```
>>> === RESTART ===
>>>
Day:  13 Apr 2012
Time: 15:50:05 PM

Day:  13 Apr 2012
Time: 15:50:07 PM

Day:  13 Apr 2012
Time: 15:50:11 PM
```

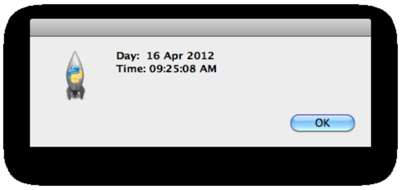
Option `command` specifies the function that is executed every time the button is clicked

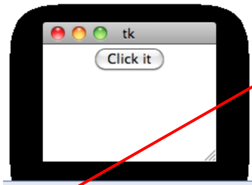
This function is called an event handler: it handles the event of clicking this particular button

```
from tkinter import Tk, Button
from time import strftime, localtime

def clicked():
    'prints day and time info'
    time = strftime('Day:  %d %b %Y\nTime: %H:%M:%S %p\n',
                    localtime())
    print(time)

root = Tk()
button = Button(root,
                text='Click it',
                command=clicked)
button.pack()
root.mainloop()
```

clickit.py

Widget Button


Day: 16 Apr 2012
Time: 09:25:08 AM



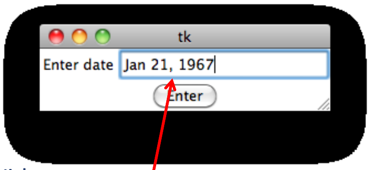
```
>>> === RESTART ===
>>>
Day:  13 Apr 2012
Time: 15:50:05 PM

Day:  13 Apr 2012
Time: 15:50:07 PM

Day:  13 Apr 2012
Time: 15:50:07 PM
```

Suppose we want the date and time to be printed in a window, rather than in the shell

```
from tkinter import Tk, Button
from time import strftime, localtime
from tkinter.messagebox import showinfo

def clicked():
    'prints day and time info'
    time = strftime('Day:  %d %b %Y\nTime: %H:%M:%S %p\n',
                    localtime())
    showinfo(message = time)

root = Tk()
button = Button(root,
                text='Click it',
                command=clicked)
button.pack()
root.mainloop()
```

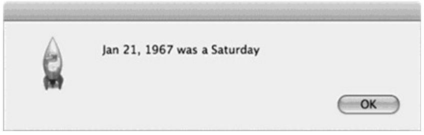
clickit.py

Widget `Entry`



tk

Enter date Jan 21, 1967



Jan 21, 1967 was a Saturday

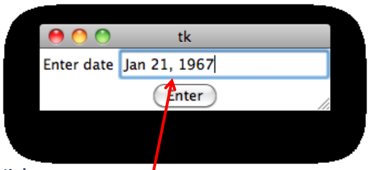
Widget `Entry` represents the single-line text entry/display form

To illustrate it, let's build an app that takes a date and prints the day of the week corresponding to the date

Event handler `compute()` should:

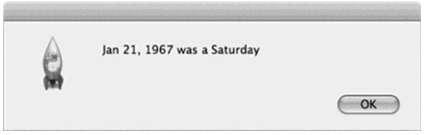
1. Read the date from entry `dateEnt`
2. Compute the weekday corresponding to the date
3. Display the weekday message in a pop-up window
4. Erase the date from entry `dateEnt` (to make it easier to enter another date)

Widget `Entry`



tk

Enter date Jan 21, 1967

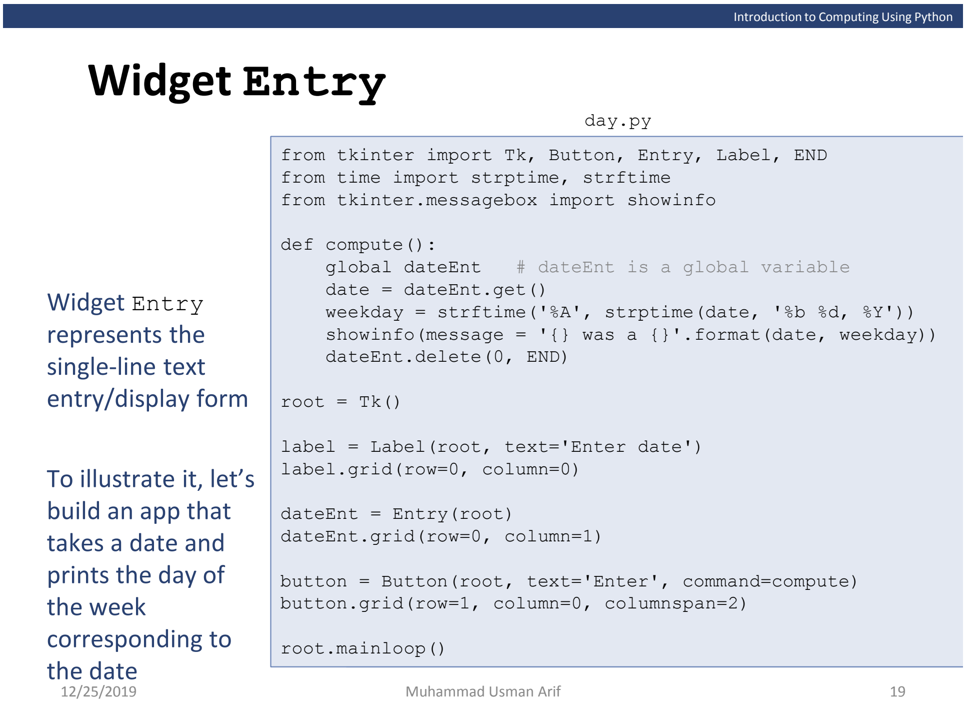


Jan 21, 1967 was a Saturday

Widget `Entry` represents the single-line text entry/display form

To illustrate it, let's build an app that takes a date and prints the day of the week corresponding to the date

```
def compute():
    # implement this

root = Tk()

label = Label(root, text='Enter date')
label.grid(row=0, column=0)

dateEnt = Entry(root)
dateEnt.grid(row=0, column=1)

button = Button(root, text='Enter', command=compute)
button.grid(row=1, column=0, columnspan=2)

root.mainloop()
```



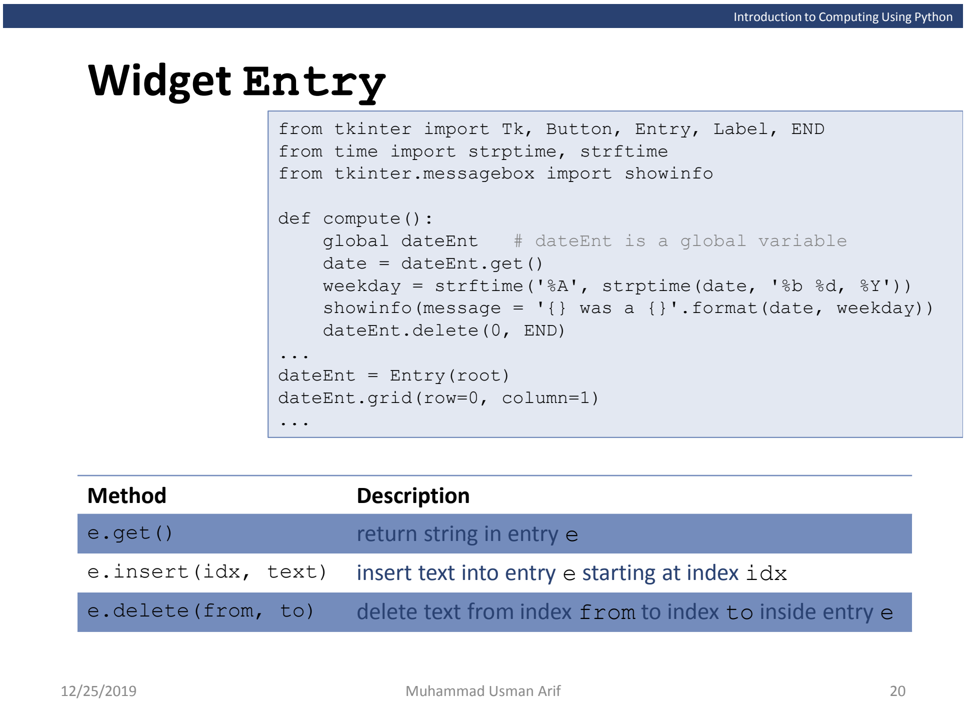
Widget Entry

Widget Entry represents the single-line text entry/display form

To illustrate it, let’s build an app that takes a date and prints the day of the week corresponding to the date

day.py

```
from tkinter import Tk, Button, Entry, Label, END
from time import strptime, strftime
from tkinter.messagebox import showinfo

def compute():
    global dateEnt   # dateEnt is a global variable
    date = dateEnt.get()
    weekday = strftime('%A', strptime(date, '%b %d, %Y'))
    showinfo(message = '{} was a {}'.format(date, weekday))
    dateEnt.delete(0, END)

root = Tk()

label = Label(root, text='Enter date')
label.grid(row=0, column=0)

dateEnt = Entry(root)
dateEnt.grid(row=0, column=1)

button = Button(root, text='Enter', command=compute)
button.grid(row=1, column=0, columnspan=2)

root.mainloop()
```

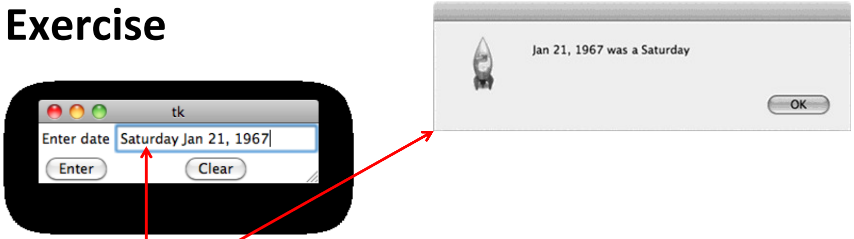


Widget Entry

```
from tkinter import Tk, Button, Entry, Label, END
from time import strptime, strftime
from tkinter.messagebox import showinfo

def compute():
    global dateEnt   # dateEnt is a global variable
    date = dateEnt.get()
    weekday = strftime('%A', strptime(date, '%b %d, %Y'))
    showinfo(message = '{} was a {}'.format(date, weekday))
    dateEnt.delete(0, END)
...
dateEnt = Entry(root)
dateEnt.grid(row=0, column=1)
...
```

Method	Description
e.get()	return string in entry e
e.insert(idx, text)	insert text into entry e starting at index idx
e.delete(from, to)	delete text from index from to index to inside entry e

Exercise



Saturday Jan 21, 1967

Jan 21, 1967 was a Saturday

Modify the app so that instead of displaying the weekday message in a separate pop-up window, insert it in front of the date in the entry box.

Also add a button labeled “Clear” that erases the entry box.

Exercise

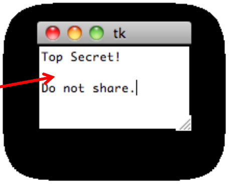
```
from tkinter import Tk, Button, Entry, Label, END
from time import strptime, strftime
from tkinter.messagebox import showinfo

def compute():
    global dateEnt    # dateEnt is a global variable
    date = dateEnt.get()
    weekday = strftime('%A', strptime(date, '%b %d, %Y'))
    dateEnt.insert(0, weekday + ' ')

def clear():
    global dateEnt    # dateEnt is a global variable
    dateEnt.delete(0, END)

root = Tk()

label = Label(root, text='Enter date')
label.grid(row=0, column=0)

dateEnt = Entry(root)
dateEnt.grid(row=0, column=1)

button = Button(root, text='Enter', command=compute)
button.grid(row=1, column=0)

button = Button(root, text='Clear', command=clear)
button.grid(row=1, column=1)

root.mainloop()
```

Widget Text

We use a `Text` widget to develop an application that looks like a text editor, but “secretly” records and prints every keystroke the user types

```
>>>
char = T
char = o
char = p
char = space
char = S
char = e
char = c
char = r
char = e
char = t
char = exclam
char = Return
char = Return
char = D
char = o …….
```

Widget `Text` represents the multi-line text entry/display form



Top Secret!

Do not share.

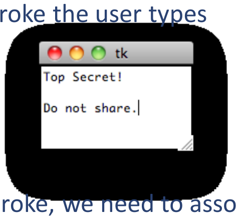
Like widget `Entry`, it supports methods `get()`, `insert()`, `delete()`

- except that the index has the format `row.column`

Method	Description
`t.get(from, to)`	return text from index `from` to index `to` in text entry `t`
`t.insert(idx, text)`	insert `text` into text entry `t` starting at index `idx`
`t.delete(from, to)`	delete text from index `from` to index `to` inside text entry `t`

Widget Text

We use a `Text` widget to develop an application that looks like a text editor, but “secretly” records and prints every keystroke the user types



Top Secret!

Do not share.

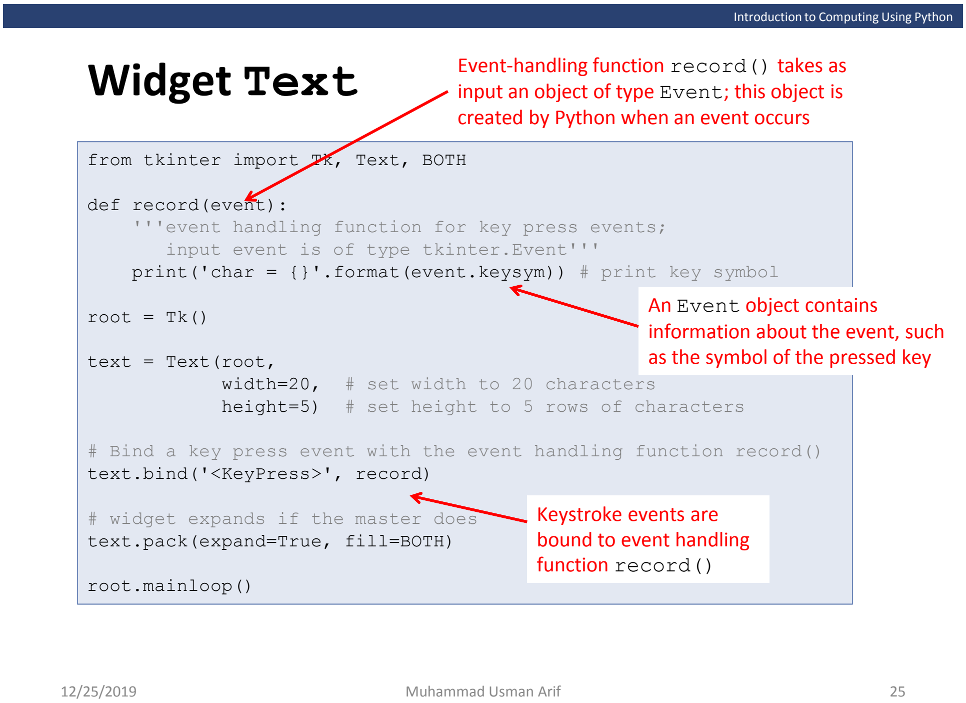
```
>>>
char = T
char = o
char = p
char = space
char = S
char = e
char = c
char = r
char = e
char = t
char = exclam
char = Return
char = Return
char = D
char = o
char = space
char = n
char = o
char = t
char = space
char = s
char = h
char = a
char = r
char = e
char = period
```

In order to record every keystroke, we need to associate an event-handling function with keystrokes

Widget method `bind()` method “binds” (i.e., associates) an event type to an event handler. For example

```
text.bind('<KeyPress>', record)
```

binds a keystroke, described with string `'<KeyPress>'`, within widget `text` to event handler `record()`



Introduction to Computing Using Python

Widget Text

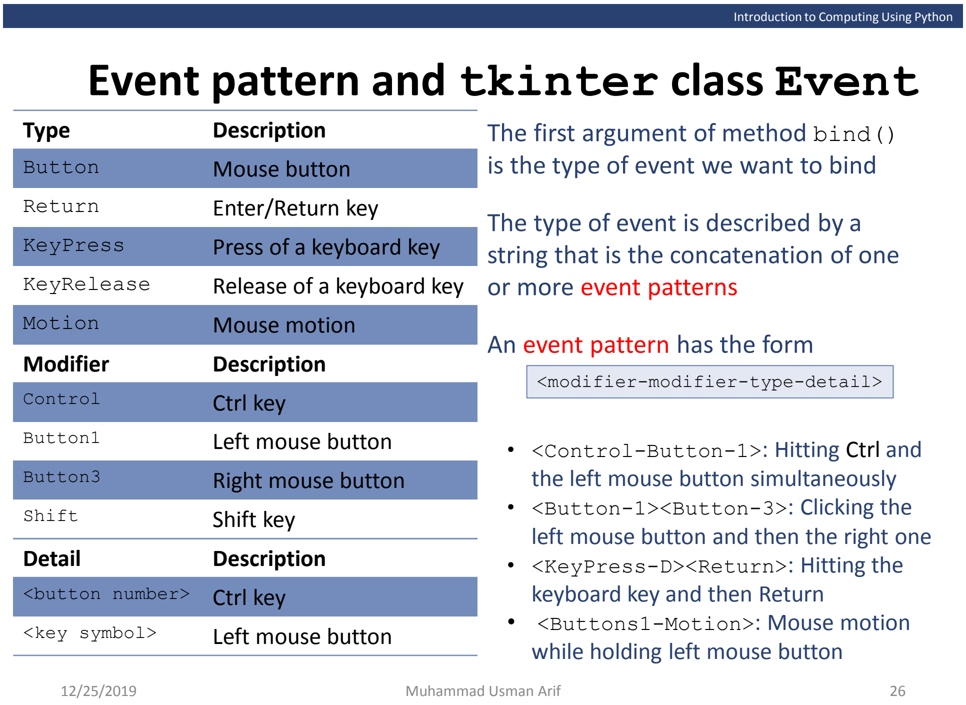
Event-handling function record() takes as input an object of type Event; this object is created by Python when an event occurs

```
from tkinter import Tk, Text, BOTH

def record(event):
    '''event handling function for key press events;
       input event is of type tkinter.Event'''
    print('char = {}'.format(event.keysym)) # print key symbol

root = Tk()

text = Text(root,
            width=20,  # set width to 20 characters
            height=5)  # set height to 5 rows of characters

# Bind a key press event with the event handling function record()
text.bind('<KeyPress>', record)

# widget expands if the master does
text.pack(expand=True, fill=BOTH)

root.mainloop()
```

An Event object contains information about the event, such as the symbol of the pressed key

Keystroke events are bound to event handling function record()

12/25/2019 Muhammad Usman Arif 25



Introduction to Computing Using Python

Event pattern and tkinter class Event

Type	Description
Button	Mouse button
Return	Enter/Return key
KeyPress	Press of a keyboard key
KeyRelease	Release of a keyboard key
Motion	Mouse motion
Modifier	**Description**
Control	Ctrl key
Button1	Left mouse button
Button3	Right mouse button
Shift	Shift key
Detail	**Description**
<button number>	Ctrl key
<key symbol>	Left mouse button

The first argument of method bind() is the type of event we want to bind

The type of event is described by a string that is the concatenation of one or more event patterns

An event pattern has the form

```
<modifier-modifier-type-detail>
```

- <Control-Button-1>: Hitting Ctrl and the left mouse button simultaneously
- <Button-1><Button-3>: Clicking the left mouse button and then the right one
- <KeyPress-D><Return>: Hitting the keyboard key and then Return
- <Buttons1-Motion>: Mouse motion while holding left mouse button

12/25/2019 Muhammad Usman Arif 26

Event pattern and `tkinter` class `Event`

The second argument of method `bind()` is the event handling function

The event handling function must be defined to take exactly one argument, an object of type `Event`, a class defined in `tkinter`

When an event occurs, Python will create an object of type `Event` associated with the event and then call the event-handling function with the `Event` object passed as the single argument

An `Event` object has many attributes that store information about the event

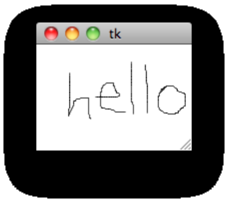
Attribute	Event Type	Description
`num`	ButtonPress, ButtonRelease	Mouse button pressed
`time`	all	Time of event
`x`	all	x-coordinate of mouse
`y`	all	y-coordinate of mouse
`Keysym`	KeyPress, KeyRelease	Key pressed as string
`keysym_num`	KeyPress, KeyRelease	Key pressed as Unicode number

Exercise

In the original day.py program, the user has to click button “Enter” after typing a date in the entry box. Requiring the user to use the mouse right after typing his name using the keyboard is an inconvenience. Modify the program day.py to allow the user just to press the Enter/Return keyboard key instead of clicking the button “Enter”.

Widget Canvas

Widget Canvas represents a drawing board in which lines and other geometrical objects can be drawn



We illustrate widget Canvas by developing a pen drawing app

- the user starts the drawing of the curve by pressing the left mouse button
- the user then draws the curve by moving the mouse, while still pressing the left mouse button

Widget Canvas

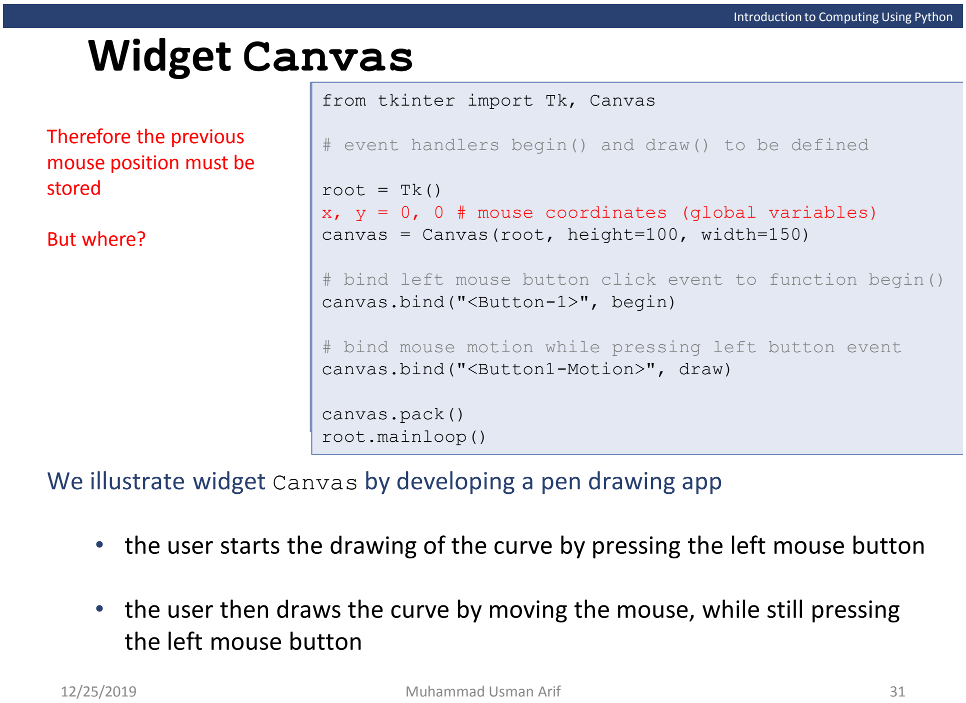
Every time the mouse is moved while pressing the left mouse button, the handler draw() is called with an Event object storing the new mouse position.

To continue drawing the curve, we need to connect this new mouse position to the previous one with a straight line.

```
from tkinter import Tk, Canvas

# event handlers begin() and draw() to be defined

root = Tk()
canvas = Canvas(root, height=100, width=150)

# bind left mouse button click event to function begin()
canvas.bind("<Button-1>", begin)

# bind mouse motion while pressing left button event
canvas.bind("<Button1-Motion>", draw)

canvas.pack()
root.mainloop()
```

We illustrate widget Canvas by developing a pen drawing app

- the user starts the drawing of the curve by pressing the left mouse button
- the user then draws the curve by moving the mouse, while still pressing the left mouse button



Widget Canvas

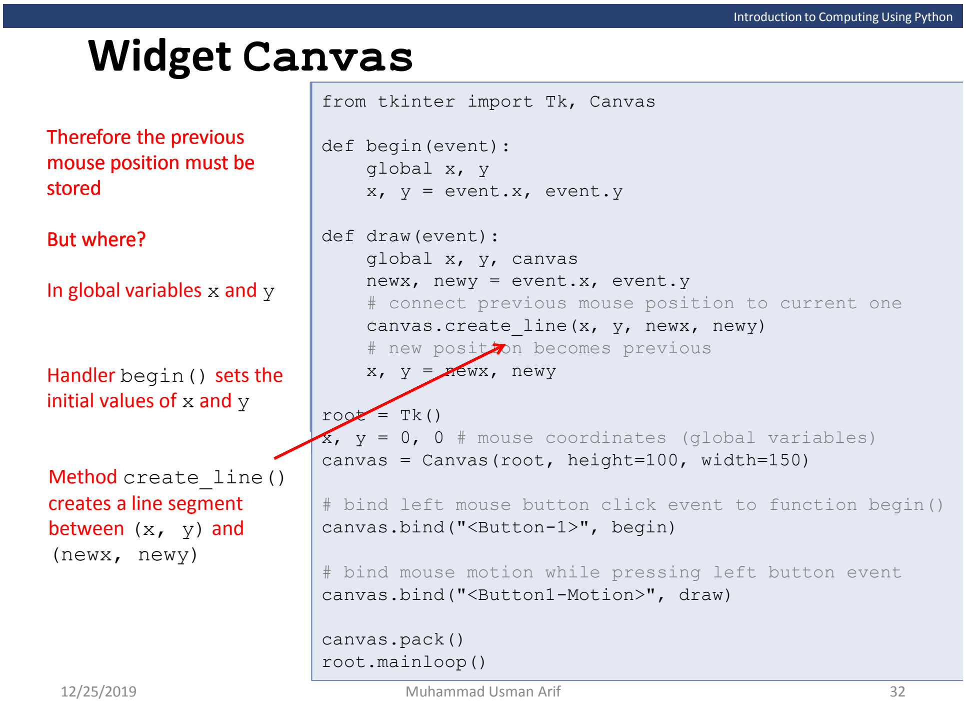
Therefore the previous mouse position must be stored

But where?

```
from tkinter import Tk, Canvas

# event handlers begin() and draw() to be defined

root = Tk()
x, y = 0, 0 # mouse coordinates (global variables)
canvas = Canvas(root, height=100, width=150)

# bind left mouse button click event to function begin()
canvas.bind("<Button-1>", begin)

# bind mouse motion while pressing left button event
canvas.bind("<Button1-Motion>", draw)

canvas.pack()
root.mainloop()
```

We illustrate widget Canvas by developing a pen drawing app

- the user starts the drawing of the curve by pressing the left mouse button
- the user then draws the curve by moving the mouse, while still pressing the left mouse button



Widget Canvas

Therefore the previous mouse position must be stored

But where?

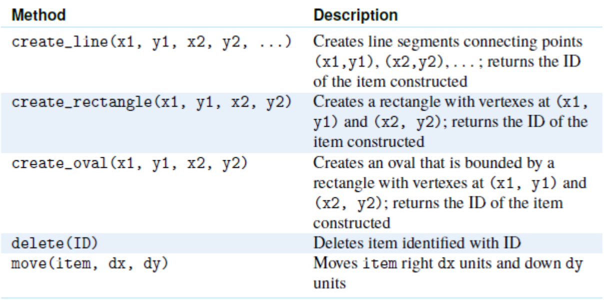
In global variables x and y

Handler begin() sets the initial values of x and y

Method create_line() creates a line segment between (x, y) and (newx, newy)

```
from tkinter import Tk, Canvas

def begin(event):
    global x, y
    x, y = event.x, event.y

def draw(event):
    global x, y, canvas
    newx, newy = event.x, event.y
    # connect previous mouse position to current one
    canvas.create_line(x, y, newx, newy)
    # new position becomes previous
    x, y = newx, newy

root = Tk()
x, y = 0, 0 # mouse coordinates (global variables)
canvas = Canvas(root, height=100, width=150)

# bind left mouse button click event to function begin()
canvas.bind("<Button-1>", begin)

# bind mouse motion while pressing left button event
canvas.bind("<Button1-Motion>", draw)

canvas.pack()
root.mainloop()
```

Widget Canvas

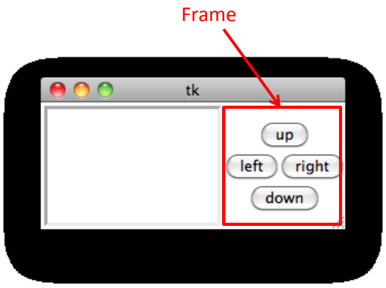


Method	Description
create_line(x1, y1, x2, y2, ...)	Creates line segments connecting points (x1,y1), (x2,y2),...; returns the ID of the item constructed
create_rectangle(x1, y1, x2, y2)	Creates a rectangle with vertexes at (x1, y1) and (x2, y2); returns the ID of the item constructed
create_oval(x1, y1, x2, y2)	Creates an oval that is bounded by a rectangle with vertexes at (x1, y1) and (x2, y2); returns the ID of the item constructed
delete(ID)	Deletes item identified with ID
move(item, dx, dy)	Moves item right dx units and down dy units

Some Canvas **methods.** Only a few methods of tkinter widget class Canvas are listed. Every object drawn in the canvas has a unique ID (which happens to be an integer).

Widget `Frame`

Widget `Frame` is a key widget whose primary purpose is to serve as the master of other widgets and help define a hierarchical structure of the GUI and its geometry



We illustrate widget `Frame` by developing an *Etch-A-Sketch* drawing app

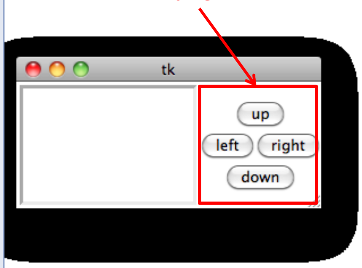
- Pressing a button moves the pen 10 pixels in the indicated direction

To facilitate the specification of the geometry of the GUI widgets, we use a `Frame` widget to be the master of the 4 buttons

Widget `Frame`

```
from tkinter import Tk, Canvas, Frame, Button,
SUNKEN, LEFT, RIGHT

# event handlers to be defined here

root = Tk()
canvas = Canvas(root, height=100, width=150,
                relief=SUNKEN, borderwidth=3)
canvas.pack(side=LEFT)

box = Frame(root) # frame to hold the 4 buttons
box.pack(side=RIGHT)

# buttons have Frame widget as their master
button = Button(box, text='up', command=up)
button.grid(row=0, column=0, columnspan=2)
button = Button(box, text='left', command=left)
button.grid(row=1, column=0)
button = Button(box, text='right', command=right)
button.grid(row=1, column=1)
button = Button(box, text='down', command=down)
button.grid(row=2, column=0, columnspan=2)

x, y = 50, 75     # initial pen position
root.mainloop()
```

Frame



Exercise

```
def up():
    'move pen up 10 pixels'
    global y, canvas
    canvas.create_line(x, y, x, y-10)
    y -= 10

def down():
    'move pen down 10 pixels'
    global y, canvas
    canvas.create_line(x, y, x, y+10)
    y += 10

def left():
    'move pen left 10 pixels'
    global x, canvas
    canvas.create_line(x, y, x-10, y)
    x -= 10

def right():
    'move pen right 10 pixels'
    global x, canvas
    canvas.create_line(x, y, x+10, y)
    x += 10
```

Implement the 4 event handlers

Note: the x coordinates increase from left to right, while the y coordinates increase from top to bottom