# 21CSC203P -Advanced Programming Practice

## Unit-5

## FORMAL AND SYMBOLIC PROGRAMMING PARADIGM

## Outline of the Presentation

1. Automata Based programming Paradigm
    - --Finite Automata – DFA and NFA
    - --Implementing using Automaton Library
2. Symbolic Programming Paradigm
    - --Algebraic manipulations and calculus
    - -- Sympy Library
3. Event Programming Paradigm
    - --Event Handler;
    - --Trigger functions and Events
    - --Tkinter Library

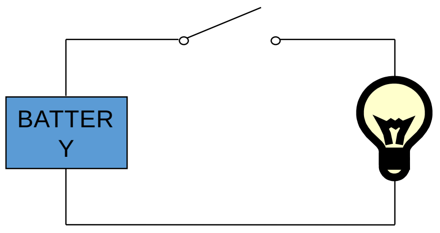
## Introduction

Automata-based programming is a programming paradigm in which the program or its part is thought of as a model of a finite state machine or any other formal automation.

**What is Automata Theory?**

- Automata theory is the study of abstract computational devices
- Abstract devices are (simplified) models of real computations
- Computations happen everywhere: On your laptop, on your cell phone, in nature, …

**Example:**



input: switch

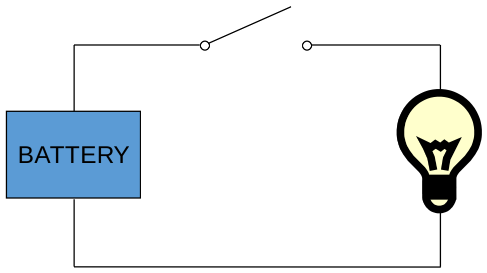
output: light bulb

actions: flip switch

states: on, off

## Simple Computer

**Example:**

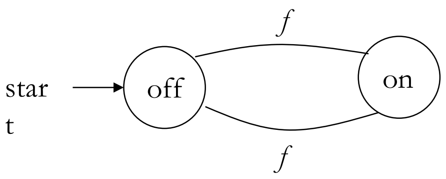


input: switch

output: light bulb

actions: flip switch

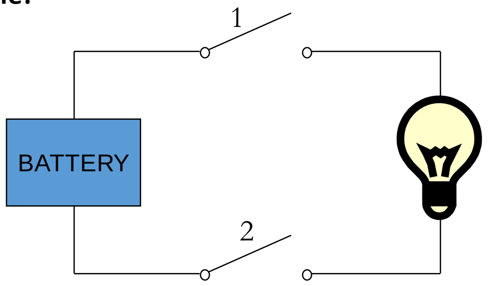
states: on, off

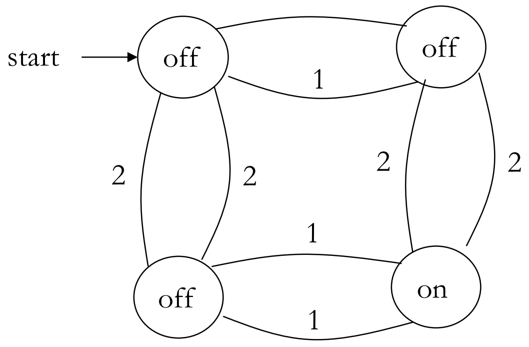


bulb is on if and only if there was an odd number of flips

## Another “computer”

**Example:**





inputs: switches 1 and 2

actions: 1 for “flip switch 1”

actions: 2 for “flip switch 2”

states: on, off

## Types of Automata

| | |
|---|---|
| finite automata | Devices with a finite amount of memory. Used to model “small” computers. |
| push-down automata | Devices with infinite memory that can be accessed in a restricted way. Used to model parsers, etc. |
| Turing Machines | Devices with infinite memory. Used to model any computer. |

## Alphabets and strings

A common way to talk about words, number, pairs of words, etc. is by representing them as strings. To define strings, we start with an alphabet

An alphabet is a finite set of symbols.

**Examples:** $\Sigma_1 = \{a, b, c, d, \ldots, z\}$: the set of letters in English

$\Sigma_2 = \{0, 1, \ldots, 9\}$: the set of (base 10) digits

$\Sigma_3 = \{a, b, \ldots, z, \#\}$: the set of letters plus the special symbol #

$\Sigma_4 = \{\,(,\,)\,\}$: the set of open and closed brackets

## Strings

A string over alphabet Σ is a finite sequence of symbols in Σ.

The empty string will be denoted by ∈

**Examples:**

abfbz is a string over $\Sigma_1 = \{a, b, c, d, \ldots, z\}$

9021 is a string over $\Sigma_2 = \{0, 1, \ldots, 9\}$

ab#bc is a string over $\Sigma_3 = \{a, b, \ldots, z, \#\}$

))()(() is a string over $\Sigma_4 = \{ (, ) \}$

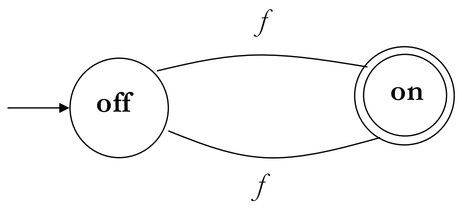
## Languages

A language is a set of strings over an alphabet.

Languages can be used to describe problems with "yes/no" answers, for example:

$L_1 =$ The set of all strings over $\Sigma_1$ that contain the substring "SRM"
$L_2 =$ The set of all strings over $\Sigma_2$ that are divisible by 7 $= \{7, 14, 21, \ldots\}$
$L_3 =$ The set of all strings of the form $\mathrm{s\#s}$ where s is any string over $\{\mathrm{a, b, \ldots, z}\}$
$L_4 =$ The set of all strings over $\Sigma_4$ where every ( can be matched with a subsequent )

## Finite Automata



There are states off and on, the automaton starts in off and tries to reach the “good state” on

What sequences of fs lead to the good state?

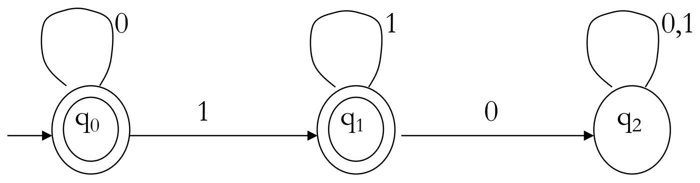
Answer: {f, fff, fffff, …} = {f n: n is odd}

This is an example of a deterministic finite automaton over alphabet {f}

## Deterministic finite automata

- A deterministic finite automaton (DFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where
  - $Q$ is a finite set of states
  - $\Sigma$ is an alphabet
  - $\delta: Q \times \Sigma \rightarrow Q$ is a transition function
  - $q_0 \in Q$ is the initial state
  - $F \subseteq Q$ is a set of accepting states (or final states).
- In diagrams, the accepting states will be denoted by double loops

## Example



alphabet $\Sigma = \{0, 1\}$
state $\mathcal{Q} = \{q_0, q_1, q_2\}$
initial state $q_0$
accepting states $F = \{q_0, q_1\}$

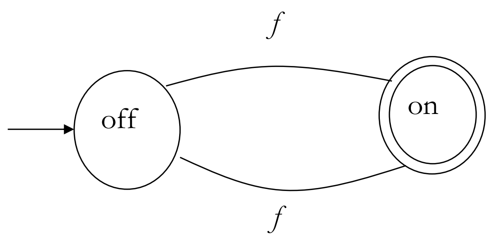
transition function δ:

| states \ inputs | 0 | 1 |
|---|---|---|
| $q_0$ | $q_0$ | $q_1$ |
| $q_1$ | $q_2$ | $q_1$ |
| $q_2$ | $q_2$ | $q_2$ |

## Language of a DFA

The language of a DFA $(Q, \Sigma, \delta, q_0, F)$ is the set of all strings over $\Sigma$ that, starting from $q_0$ and following the transitions as the string is read left to right, will reach some accepting state.

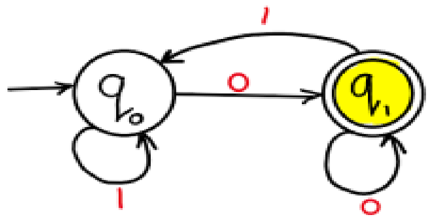
*M*:



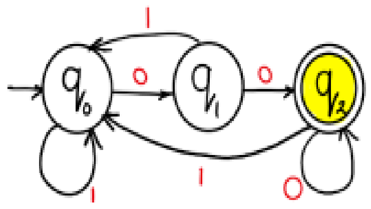
- Language of $M$ is $\{f, fff, fffff, \dots\} = \{f^n: n \text{ is odd}\}$

## Example of DFA

DFA for the language accepting strings ending with ‘0’ over input alphabets ∑={0, 1} ?

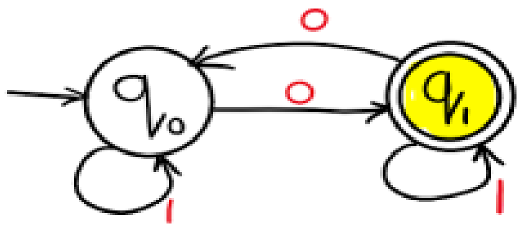


DFA for the language accepting strings ending with ‘00’ over input alphabets ∑={0, 1} ?



## Example of DFA

Draw a DFA for the language accepting strings containing even number of total zeros over input alphabets ∑ = {0, 1} ?



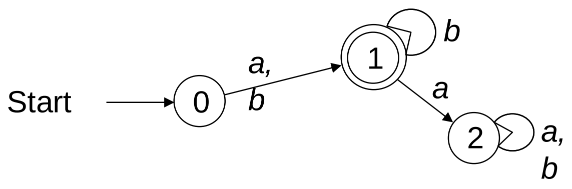
## Example of DFA using Python

```
from automata.fa.dfa import DFA
# DFA which matches all binary strings ending in an odd number of '1's
dfa = DFA(
    states={'q0', 'q1', 'q2'},
    input_symbols={'0', '1'},
    transitions={
        'q0': {'0': 'q0', '1': 'q1'},
        'q1': {'0': 'q0', '1': 'q2'},
        'q2': {'0': 'q2', '1': 'q1'}
    },
    initial_state='q0',
    final_states={'q1'}
)
dfa.read_input('01')   # answer is  'q1‘
dfa.read_input('011')  # answer is error
print(dfa.read_input_stepwise('011'))
Answer # yields:
# 'q0‘     # 'q0‘     # 'q1'
# 'q2‘     # 'q1'
```

```
if dfa.accepts_input('011'):
    print('accepted')
else:
    print('rejected')
```

## Table Representation of a DFA

A DFA over A can be represented by a transition function T : States X A -> States, where T(i, a) is the state reached from state i along the edge labelled a, and we mark the start and final states. For example, the following figures show a DFA and its transition table.



|  | $T$ | $a$ | $b$ |
|---|---|---|---|
| start | 0 | 1 | 1 |
| final | 1 | 2 | 1 |
|  | 2 | 2 | 2 |

## Sample Exercises - DFA

1. Write a automata code for the Language that accepts all and only those strings that contain 001
2. Write a automata code for L(M) ={ w | w has an even number of 1s}
3. Write a automata code for L(M) ={0,1}*
4. Write a automata code for L(M)=$a + aa^*b$.
5. Write a automata code for L(M)={$(ab)^n \mid n \in$ N}
6. Write a automata code for Let Σ = {0, 1}.

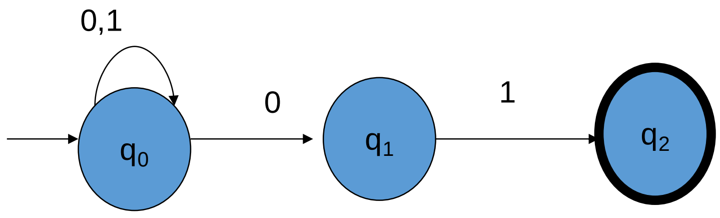
   Given DFAs for {}, {ε}, $\Sigma^*$, and $\Sigma^+$.

## NDFA

- A nondeterministic finite automaton M is a five-tuple M = (Q, Σ, δ, $q_0$, F), where:
  - Q is a finite set of states of M
  - Σ is the finite input alphabet of M
  - δ: Q × Σ → power set of Q, is the state transition function mapping a state-symbol pair to a subset of Q
  - $q_0$ is the start state of M
  - F ⊆ Q is the set of accepting states or final states of M

## Example NDFA

- NFA that recognizes the language of strings that end in 01

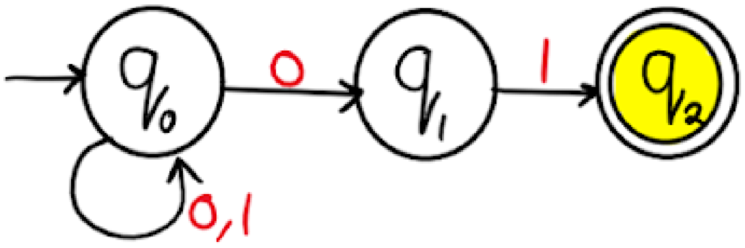


note: $\delta(q_0,0) = \{q_0,q_1\}$
$\delta(q_1,0) = \{\}$

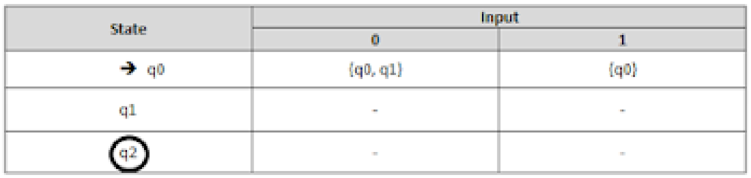
Exercise: Draw the complete transition table for this NFA

## Example NDFA

Consider the NFA that accepts all string ending with 01.



Transition diagram



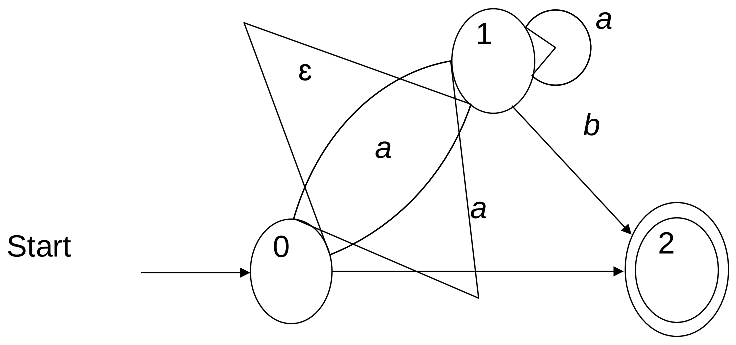
| State | Input | |
|---|---|---|
| | 0 | 1 |
| → q0 | {q0, q1} | {q0} |
| q1 | - | - |
| q2 | - | - |

Transition table

## NDFA

A nondeterministic finite automaton (NFA) over an alphabet A is similar to a DFA except that epislon-edges are allowed, there is no requirement to emit edges from a state, and multiple edges with the same letter can be emitted from a state.

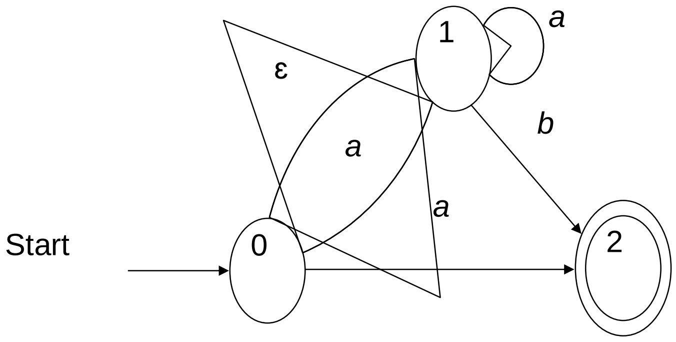
**Example**. The following NFA recognizes the language of a + aa*b + a*b.



## NDFA

**Table representation of NFA**

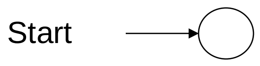
An NFA over A can be represented by a function T : States × A ∪ {L} → power(States), where T(i, a) is the set of states reached from state i along the edge labeled a, and we mark the start and final states. The following figure shows the table for the preceding NFA.



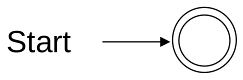
| | $T$ | $a$ | $b$ | e |
|---|---|---|---|---|
| start | 0 | {1, 2} | ∅ | {1} |
| | 1 | {1} | {2} | ∅ |
| final | 2 | ∅ | ∅ | ∅ |

## Examples

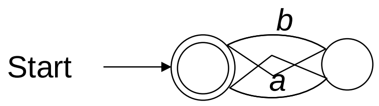
***Solutions***: (a):



(b):

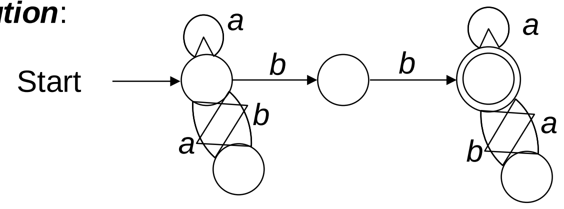


(c):



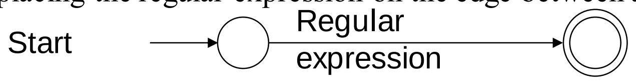
Find an NFA to recognize the language ($a$ + $ba$)*$bb$($a$ + $ab$)*.

***A solution***:

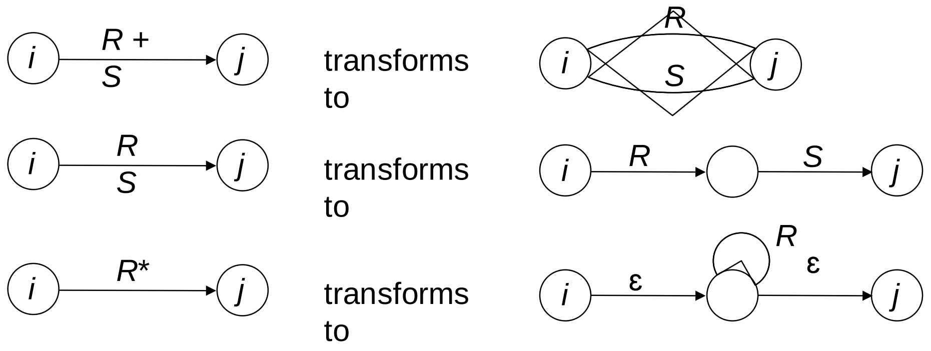


## Examples

Algorithm: *Transform a Regular Expression into a Finite Automaton*
Start by placing the regular expression on the edge between a start and final state:

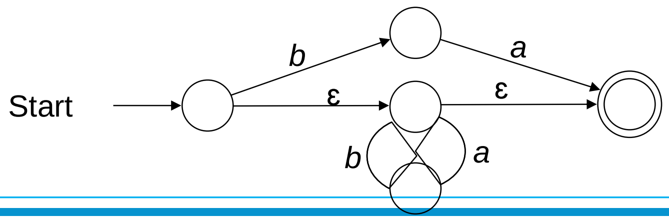


Apply the following rules to obtain a finite automaton after erasing any ∅-edges.



*Quiz.* Use the algorithm to construct a finite automaton for $(ab)^* + ba$.

***Answer*:**



## Example of NFA using Python

```
from automata.fa.nfa import NFA
# NFA which matches strings beginning with 'a', ending with 'a', and
containing
# no consecutive 'b's
nfa = NFA(
    states={'q0', 'q1', 'q2'},
    input_symbols={'a', 'b'},
    transitions={
        'q0': {'a': {'q1'}},
        # Use '' as the key name for empty string (lambda/epsilon)
transitions
        'q1': {'a': {'q1'}, '': {'q2'}},
        'q2': {'b': {'q0'}}
    },
    initial_state='q0',
    final_states={'q1'}
)
```

```
nfa.read_input('aba')
ANSWER :{'q1', 'q2'}

nfa.read_input('abba')
ANSWR: ERROR


nfa.read_input_stepwise('aba')

if nfa.accepts_input('aba'):
    print('accepted')
else:
    print('rejected')
ANSWER: ACCEPTED
nfa.validate()
ANSWR: TRUE
```

## Sample Exercises - NFA

1. Write a automata code for the Language that accepts all end with 01
2. Write a automata code for L(M)= $a + aa^*b + a^*b$.
3. Write a automata code for Let $\Sigma = \{0, 1\}$.

   Given NFAs for $\{\}$, $\{\varepsilon\}$, $\{(ab)^n \mid n \in \mathrm{N}\}$, which has regular expression $(ab)^*$.

## Symbolic Programming Paradigms

- **symbolic programming** is a programming paradigm in which the program can manipulate its own formulas and program components as if they were plain data.Through symbolic programming, complex processes can be developed that build other more intricate processes by combining smaller units of logic or functionality.

- symbolic programs can effectively modify themselves and appear to learn, which makes them better suited for applications such as artificial intelligence expert systems natural language processing and computer games.

- Languages that support symbolic programming include the Language LISP and Prolog

## Sympy-Symbolic Mathematics in Python

- SymPy is a Python library used to represent the symbolic mathematics

- SymPy is written entirely in Python and does not require any external libraries.
- To Install the sympy use command : `pip install sympy`

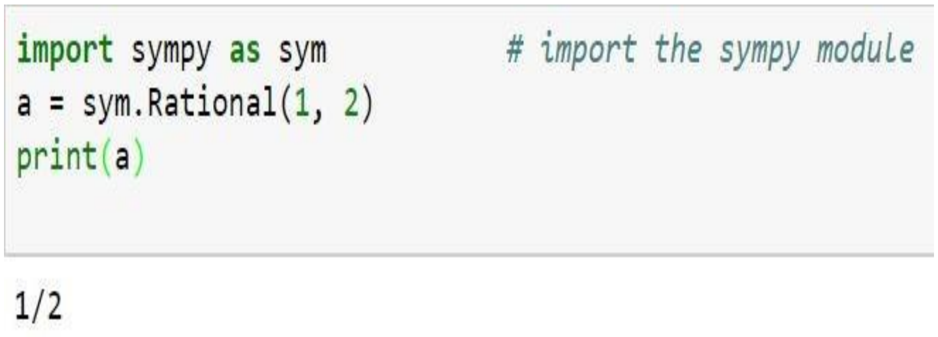
- Sympy support to perform the mathematical operations such as
- Algebraic manipulations
- Differentiation
- Integration
- Equation solving & Linear algebra

**Mathematical Operations Using Sympy:**

**To Find the rational no :**

The Rational class represents a rational number as a pair of two Integers: the numerator and the denominator.

Rational(5, 2) represents 5/2.

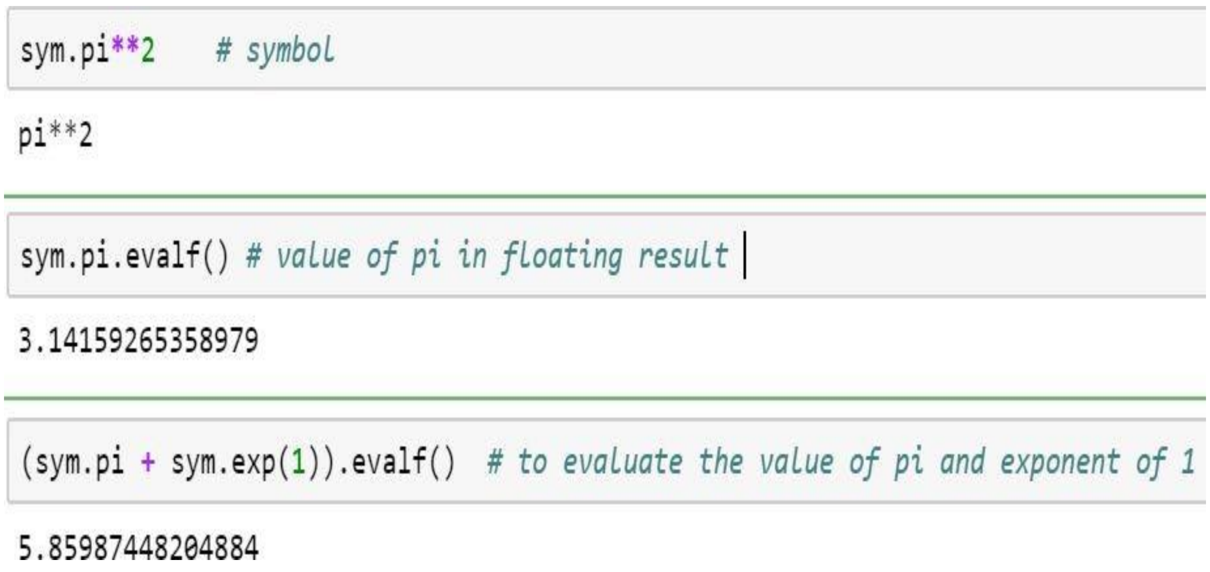


```
import sympy as sym          # import the sympy module
a = sym.Rational(1, 2)
print(a)
```

```
1/2
```

## Mathematical Operations Using Sympy:

- Some special constant e,pi,oo (Infinity) considered as symbol and evaluate the values. `evalf` evaluates the expression to a floating-point number.



```
sym.pi**2    # symbol
```

```
pi**2
```

```
sym.pi.evalf() # value of pi in floating result
```

```
3.14159265358979
```

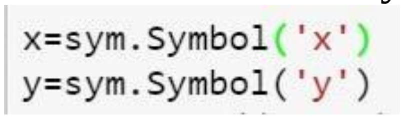
```
(sym.pi + sym.exp(1)).evalf()  # to evaluate the value of pi and exponent of 1
```

```
5.85987448204884
```

## Symbols :

In Computer Algebra Systems, in SymPy you have to declare symbolic variables explicitly:

**Symbols()** method also to declare a variable as symbol.



```
x=sym.Symbol('x')
y=sym.Symbol('y')
```

Then you can manipulate them:

```
>>> x + y + x – y
        o/p
>>>  2* x
```

## **Calculus** (differentiation and integration)

- Differentiation and integration can help us solve many types of real-world problems.
- Derivatives are met in many engineering and science problems, especially when modeling the behaviour of moving objects.

## 3.Calculus

### limit() in Python

With the help of **sympy.limit()** method, we can find the limit of any mathematical expression, e.g.,

$$\lim_{x \to a} f(x)$$

**Syntax:** limit(expression, variable, value)

**Parameters:**

**expression –** The mathematical expression on which limit opeartion is to be performed, i. e., f(x).
**variable –** It is the variable in the mathematical expression, i. e., x
**value –** It is the value to which the limit tends to, i. e., a.
**Returns:** Returns the limit of the mathematical expression under given conditions.

## limit() -method:

.

Ex:

```
# import sympy
from sympy import *
x = symbols('x')
expr = sin(x)/x;
# Use sympy.limit() method
limit_expr = limit(expr, x, 0)
print("Limit of the expression tends to 0 : {}".format(limit_expr))
```

**Output:**

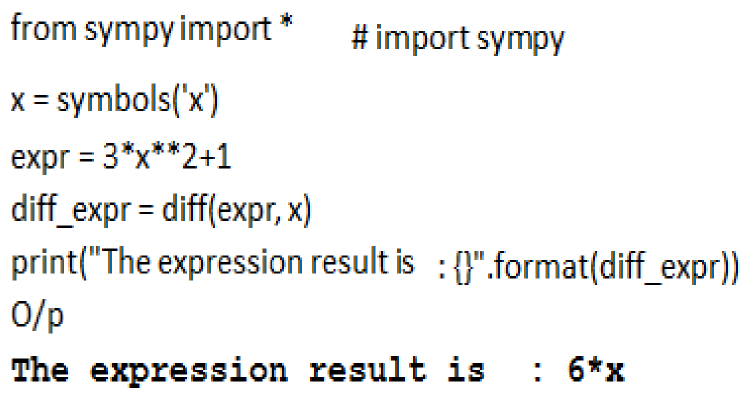
```
Limit of the expression tends to 0 : 1
```

## Differentiate in python

Differentiate any SymPy expression using **diff()** method.

.

**Syntax :** diff(func,var)



```
from sympy import *        # import sympy
x = symbols('x')
expr = 3*x**2+1
diff_expr = diff(expr, x)
print("The expression result is  : {}".format(diff_expr))
O/p
The expression result is   : 6*x
```

## Series in Python

- With the help of sympy.series() method, we can find the series of some mathematical functions and trigonometric expressions by using sympy.series() method.

  Syntax : sympy.series()

  Return : Return a series of functions

## Series in Python- Example

.

```
from sympy import * x,       # import sympy
 y = symbols('x  y')
# Use sympy.series()  method
series_fun= cos(x).series()
print(series_fun)
```

O/p:

```
1 - x**2/2 + x**4/24 + O(x**6)
```

## Integration in python

**sympy.integrate() method**

- we can find the integration of mathematical expressions in the form of variables
- syntax : sympy.integrate(expression, reference variable)
- Return : Return integration of mathematical expression using sympy.integrate() method.

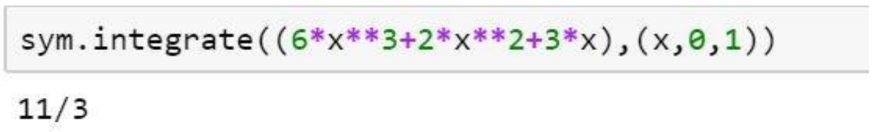
Ex1:

int_exp = sin(x)*exp(x)

intr = integrate(int_exp, x)

After Integration O/p is : exp(x)*sin(x)/2 –exp(x)*cos(x)/2

Ex2:



```
sym.integrate((6*x**3+2*x**2+3*x),(x,0,1))
```

```
11/3
```

## Algebraic manipulations

.

- SymPy is capable of performing powerful algebraic manipulations.

**Most frequently used algebraic manipulations are expand and simplify.**

**expand() in python:**

we can expand the mathematical expressions in the form of variables by using sympy.expand() method.

**Syntax : sympy.expand(expression)**

**Return : Return mathematical expression.**

## Algebraic manipulations

- **Example:**

```
import sympy as sym
x=sym.Symbol('x')
y=sym.Symbol('y')
exp =sym.expand((x+y)**2)
print(exp)
```

O/p:

```
        x**2 + 2*x*y + y**2
```

## simplify() in python

.
- We can simplify any mathematical expression using simplify() method.

  ***Syntax*****:** *simplify(expression)*

- ***Parameters:***

  ***expression*** *– It is the mathematical expression which needs to be simplified.*

```
# import sympy
from sympy import *
x = symbols('x')
expr = sin(x)**2 + cos(x)**2
# Use sympy.simplify() method
smpl = simplify(expr)
print("After Simplification : {}".format(smpl))
```

O/p - After Simplification : 1

## trigsimp() in python

. We can simplify mathematical expressions using trigonometric identities.

Syntax: trigsimp(expression)

Parameters:

- expression – It is the mathematical expression which needs to be simplified.

## trigsimp() in python

.

**Example :**

```
# import sympy
from sympy import *
x = symbols('x')
expr = sin(x)**2 + cos(x)**2
# Use sympy.trigsimp() method
smpl = trigsimp(expr)
print("After Simplification : {}".format(smpl))
```

O/p:

```
After Simplification : 1
```

## solve() in python-SymPy solving equations

.

Equations are solved with solve() or solveset().

- **Example:**

```
import sympy as sym
x = Symbol('x')
eq1 = Eq(x + 1, 3)
sol = solve(eq1, x)
print(sol)
```

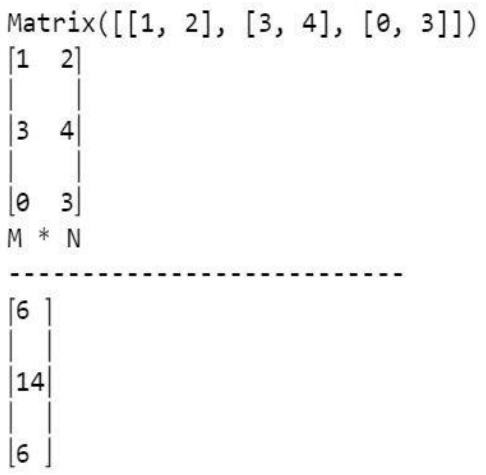
O/P

```
[2]
```

## SymPy matrixes

- In SymPy, we can work with matrixes. A matrix is a rectangular array of numbers or other mathematical objects for which operations such as addition and multiplication are defined.
- Matrixes are used in computing, engineering, or image processing.

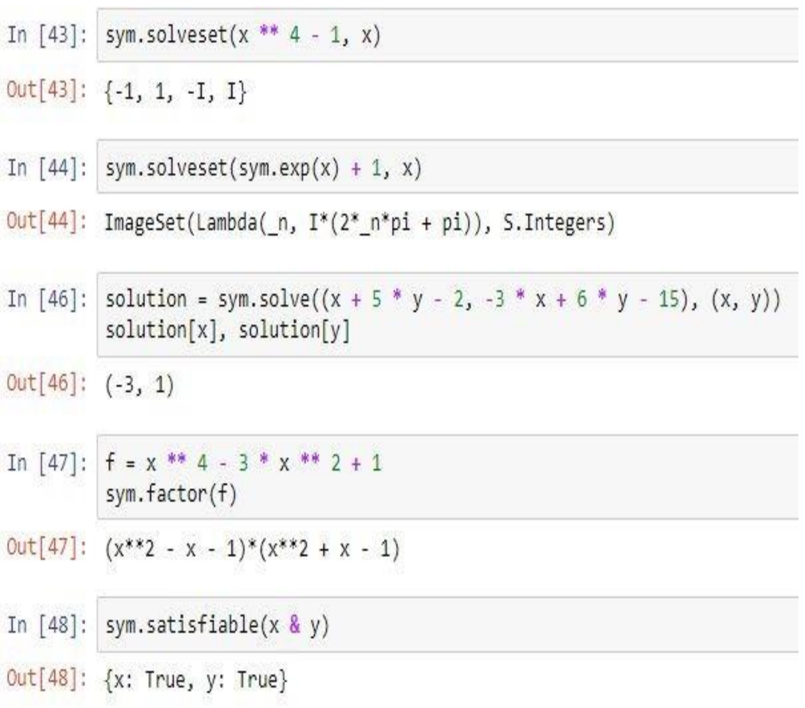
```
M = Matrix([[1, 2], [3, 4], [0, 3]])
print(M)
pprint(M)

N = Matrix([2, 2])

print("M * N")
print("---------------------------")

pprint(M*N)

O/p
--?
```



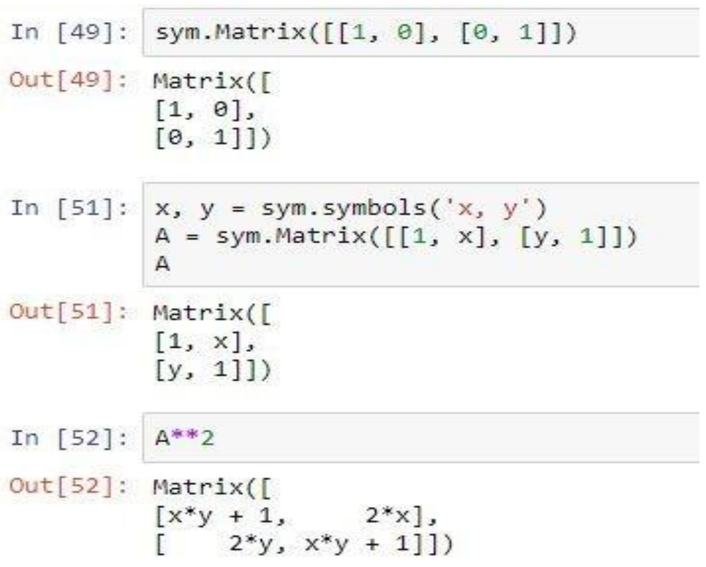
```
Matrix([[1, 2], [3, 4], [0, 3]])
⎡1  2⎤
⎢    ⎥
⎢3  4⎥
⎢    ⎥
⎣0  3⎦
M * N
---------------------------
⎡6 ⎤
⎢  ⎥
⎢14⎥
⎢  ⎥
⎣6 ⎦
```

## Example

**Example:**



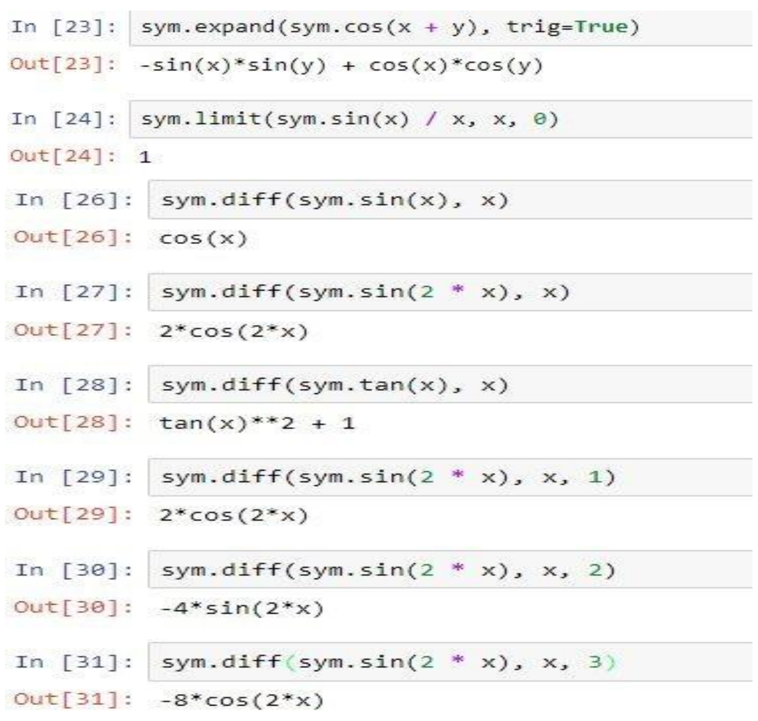
```
In [43]: sym.solveset(x ** 4 - 1, x)

Out[43]: {-1, 1, -I, I}

In [44]: sym.solveset(sym.exp(x) + 1, x)

Out[44]: ImageSet(Lambda(_n, I*(2*_n*pi + pi)), S.Integers)

In [46]: solution = sym.solve((x + 5 * y - 2, -3 * x + 6 * y - 15), (x, y))
         solution[x], solution[y]

Out[46]: (-3, 1)

In [47]: f = x ** 4 - 3 * x ** 2 + 1
         sym.factor(f)

Out[47]: (x**2 - x - 1)*(x**2 + x - 1)

In [48]: sym.satisfiable(x & y)

Out[48]: {x: True, y: True}
```



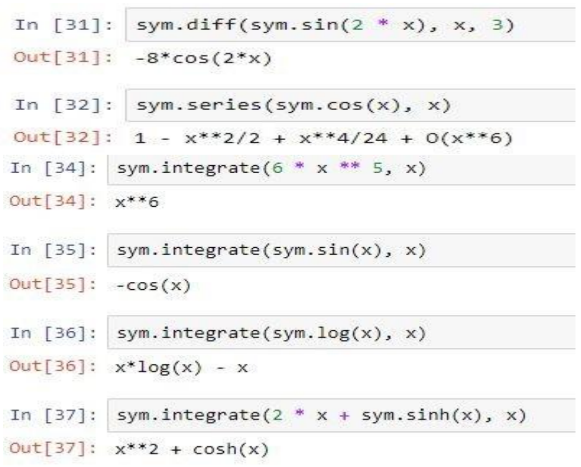
```
In [49]: sym.Matrix([[1, 0], [0, 1]])

Out[49]: Matrix([
         [1, 0],
         [0, 1]])

In [51]: x, y = sym.symbols('x, y')
         A = sym.Matrix([[1, x], [y, 1]])
         A

Out[51]: Matrix([
         [1, x],
         [y, 1]])

In [52]: A**2

Out[52]: Matrix([
         [x*y + 1,     2*x],
         [    2*y, x*y + 1]])
```

# Example

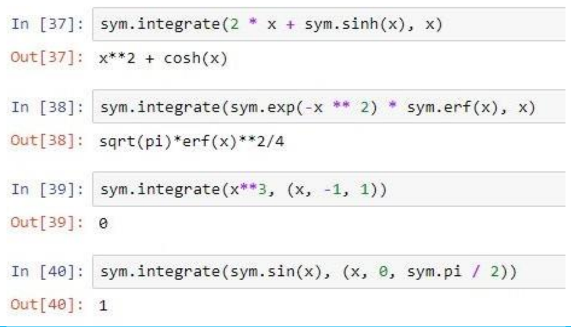
**Example:**



```
In [23]: sym.expand(sym.cos(x + y), trig=True)
Out[23]: -sin(x)*sin(y) + cos(x)*cos(y)

In [24]: sym.limit(sym.sin(x) / x, x, 0)
Out[24]: 1

In [26]: sym.diff(sym.sin(x), x)
Out[26]: cos(x)

In [27]: sym.diff(sym.sin(2 * x), x)
Out[27]: 2*cos(2*x)

In [28]: sym.diff(sym.tan(x), x)
Out[28]: tan(x)**2 + 1

In [29]: sym.diff(sym.sin(2 * x), x, 1)
Out[29]: 2*cos(2*x)

In [30]: sym.diff(sym.sin(2 * x), x, 2)
Out[30]: -4*sin(2*x)

In [31]: sym.diff(sym.sin(2 * x), x, 3)
Out[31]: -8*cos(2*x)
```



```
In [31]: sym.diff(sym.sin(2 * x), x, 3)
Out[31]: -8*cos(2*x)

In [32]: sym.series(sym.cos(x), x)
Out[32]: 1 - x**2/2 + x**4/24 + O(x**6)
In [34]: sym.integrate(6 * x ** 5, x)
Out[34]: x**6

In [35]: sym.integrate(sym.sin(x), x)
Out[35]: -cos(x)

In [36]: sym.integrate(sym.log(x), x)
Out[36]: x*log(x) - x

In [37]: sym.integrate(2 * x + sym.sinh(x), x)
Out[37]: x**2 + cosh(x)
```



```
In [37]: sym.integrate(2 * x + sym.sinh(x), x)
Out[37]: x**2 + cosh(x)

In [38]: sym.integrate(sym.exp(-x ** 2) * sym.erf(x), x)
Out[38]: sqrt(pi)*erf(x)**2/4

In [39]: sym.integrate(x**3, (x, -1, 1))
Out[39]: 0

In [40]: sym.integrate(sym.sin(x), (x, 0, sym.pi / 2))
Out[40]: 1
```

# Event Programming Paradigm

## Event Driven Programming Paradigm

- Event-driven programming is a programming paradigm in which the flow of program execution is determined by events - for example a user action such as a mouse click, key press, or a message from the operating system or another program.
- An event-driven application is designed to detect events as they occur, and then deal with them using an appropriate event-handling procedure.
- In a typical modern event-driven program, there is no discernible flow of control. The main routine is an event-loop that waits for an event to occur, and then invokes the appropriate event-handling routine.
- Event callback is a function that is invoked when something significant happens like when click event is performed by user or the result of database query is available.

## Event Driven Programming Paradigm

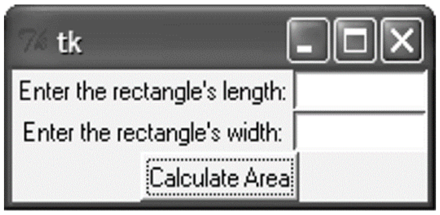
**Event Handlers:** Event handlers is a type of function or method that run a specific action when a specific event is triggered. For example, it could be a button that when user click it, it will display a message, and it will close the message when user click the button again, this is an event handler.

**Trigger Functions:** Trigger functions in event-driven programming are a functions that decide what code to run when there are a specific event occurs, which are used to select which event handler to use for the event when there is specific event occurred.

**Events:** Events include mouse, keyboard and user interface, which events need to be triggered in the program in order to happen, that mean user have to interacts with an object in the program, for example, click a button by a mouse, use keyboard to select a button and etc.

## Introduction

- A graphical user interface allows the user to interact with the operating system and other programs using graphical elements such as icons, buttons, and dialog boxes.
- GUIs popularized the use of the mouse.
- GUIs allow the user to point at graphical elements and click the mouse button to activate them.
- GUI Programs Are Event-Driven
- User determines the order in which things happen
- GUI programs respond to the actions of the user, thus they are event driven.
- The tkinter module is a wrapper around tk, which is a wrapper around tcl, which is what is used to create windows and graphical user interfaces.

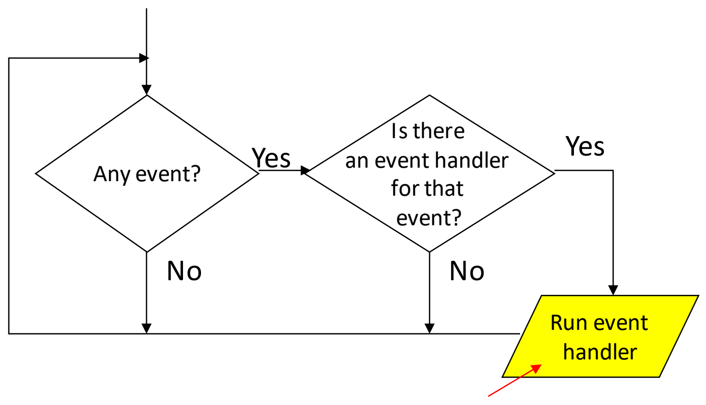


## Introduction

- A major task that a GUI designer needs to do is to determine what will happen when a GUI is invoked
- Every GUI component may generate different kinds of “events” when a user makes access to it using his mouse or keyboard
- E.g. if a user moves his mouse on top of a button, an event of that button will be generated to the Windows system
- E.g. if the user further clicks, then another event of that button will be generated (actually it is the click event)
- For any event generated, the system will first check if there is an event handler, which defines the action for that event

## Introduction

- For a GUI designer, he needs to develop the event handler to determine the action that he wants Windows to take for that event.



## GUI Using Python

- Tkinter:

    Tkinter is the Python interface to the Tk GUI toolkit shipped with Python.

- wxPython:

    This is an open-source Python interface for wx Windows

- PyQt

    This is also a Python interface for a popular cross-platform Qt GUI library.

- JPython:

    JPython is a Python port for Java which gives Python scripts seamless access to Java class libraries on the local machine

## Tkinter Programming

- Tkinter is the standard GUI library for Python.
- Creating a GUI application using Tkinter

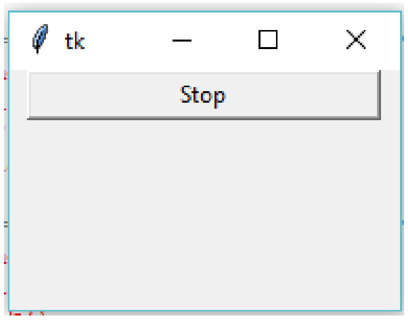
**Steps**

- Import the Tkinter module.

  *Import tKinter as tk*

- Create the GUI application main window.

  *root = tk.Tk()*



- Add one or more of the above-mentioned widgets to the GUI application.

  *button = tk.Button(root, text='Stop', width=25, command=root.destroy)*

  *button.pack()*

- Enter the main event loop to take action against each event triggered by the user.

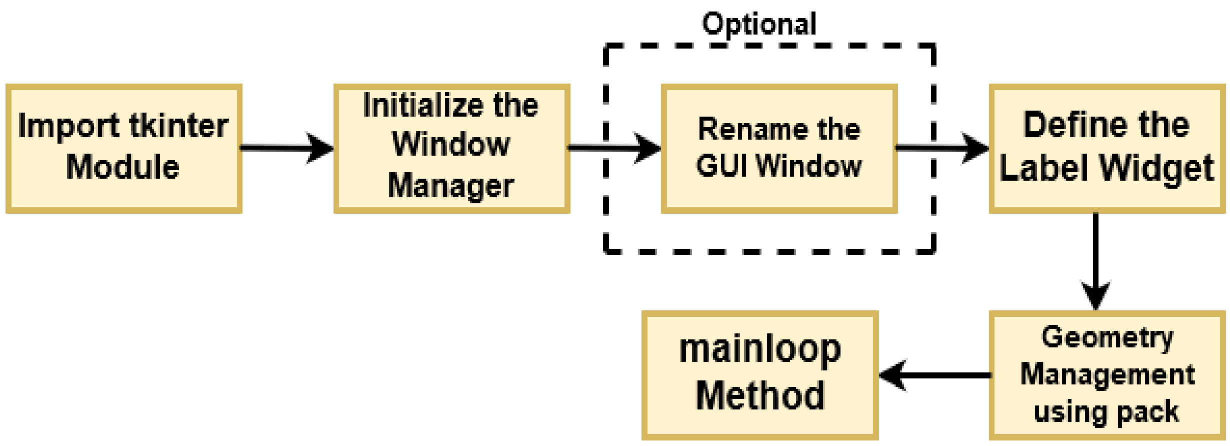
  *root.mainloop()*

## Tkinter widgets

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

| **Widget** | **Description** |
|---|---|
| Label | Used to contain text or images |
| Button | Similar to a Label but provides additional functionality for mouse overs, presses, and releases as well as keyboard activity/events |
| Canvas | Provides ability to draw shapes (lines, ovals, polygons, rectangles); can contain images or bitmaps |
| Radiobutton | Set of buttons of which only one can be "pressed" (similar to HTML radio input) |
| Checkbutton | Set of boxes of which any number can be "checked" (similar to HTML checkbox input) |
| Entry | Single-line text field with which to collect keyboard input (similar to HTML text input) |
| Frame | Pure container for other widgets |
| Listbox | Presents user list of choices to pick from |
| Menu | Actual list of choices "hanging" from a Menubutton that the user can choose from |
| Menubutton | Provides infrastructure to contain menus (pulldown, cascading, etc.) |
| Message | Similar to a Label, but displays multi-line text |
| Scale | Linear "slider" widget providing an exact value at current setting; with defined starting and ending values |
| Text | Multi-line text field with which to collect (or display) text from user (similar to HTML TextArea) |
| Scrollbar | Provides scrolling functionality to supporting widgets, i.e., Text, Canvas, Listbox, and Entry |
| Toplevel | Similar to a Frame, but provides a separate window container |

## Operation Using Tkinter Widget



## Geometry Managers

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

  **widget.pack( pack_options )**

  **options**

  - expand – When set to true, widget expands to fill any space not otherwise used in widget's parent.
  - fill – Determines whether widget fills any extra space allocated to it by the packer, or keeps its own minimal dimensions: NONE (default), X (fill only horizontally), Y (fill only vertically), or BOTH (fill both horizontally and vertically).
  - side – Determines which side of the parent widget packs against: TOP (default), BOTTOM, LEFT, or RIGHT.

## Geometry Managers

- The grid() Method – This geometry manager organizes widgets in a table-like structure in the parent widget.

  ***widget.grid( grid_options )***

  **options –**

  - Column/row – The column or row to put widget in; default 0 (leftmost column).

  - Columnspan, rowsapn– How many columns or rows to widget occupies; default 1.

  - ipadx, ipady – How many pixels to pad widget, horizontally and vertically, inside widget's borders.

  - padx, pady – How many pixels to pad widget, horizontally and vertically, outside v's borders.

## Geometry Managers

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

  **widget.place( place_options )**

  **options −**

  - anchor − The exact spot of widget other options refer to: may be N, E, S, W, NE, NW, SE, or SW, compass directions indicating the corners and sides of widget; default is NW (the upper left corner of widget)
  - bordermode − INSIDE (the default) to indicate that other options refer to the parent's inside (ignoring the parent's border); OUTSIDE otherwise ,height, width − Height and width in pixels.
  - relheight, relwidth − Height and width as a float between 0.0 and 1.0, as a fraction of the height and width of the parent widget.
  - relx, rely − Horizontal and vertical offset as a float between 0.0 and 1.0, as a fraction of the height and width of the parent widget.
  - x, y − Horizontal and vertical offset in pixels.

## Common Widget Properties

Common attributes such as sizes, colors and fonts are specified.

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

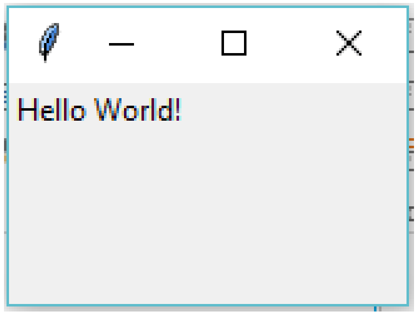
## Label Widgets

- A label is a widget that displays text or images, typically that the user will just view but not otherwise interact with. Labels are used for such things as identifying controls or other parts of the user interface, providing textual feedback or results, etc.
- **Syntax**

    *tk.Label(parent,text="message")*

***Example:***

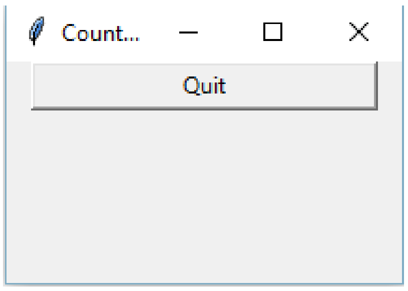
```
import tkinter as tk
root = tk.Tk()
label = tk.Label(root, text='Hello World!')
label.grid()
root.mainloop()
```



Hello World!

## Button Widgets

```
import tkinter as tk

r = tk.Tk()

r.title('Counting Seconds')

button = tk.Button(r, text='Stop', width=25, command=r.destroy)

button.pack()

r.mainloop()
```



## Button Widgets

- A button, unlike a frame or label, is very much designed for the user to interact with, and in particular, press to perform some action. Like labels, they can display text or images, but also have a whole range of new options used to control their behaviour.

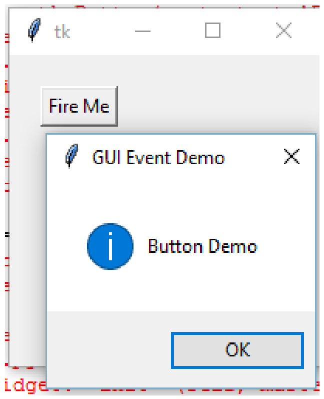
Syntax

```
button = ttk.Button(parent, text=‘ClickMe', command=submitForm)
```

## Button Widgets

*Example:*

```
import tkinter as tk
from tkinter import messagebox
def hello():
   msg = messagebox.showinfo( "GUI Event Demo","Button Demo")
root = tk.Tk()
root.geometry("200x200")
b = tk.Button(root, text='Fire Me',command=hello)
b.place(x=50,y=50)
root.mainloop()
```



## Button Widgets

- Button: To add a button in your application, this widget is used.

**Syntax :**

```
w=Button(master, text="caption" option=value)
```

- master is the parameter used to represent the parent window.
- activebackground: to set the background color when button is under the cursor.
- activeforeground: to set the foreground color when button is under the cursor.
- bg: to set he normal background color.
- command: to call a function.
- font: to set the font on the button label.
- image: to set the image on the button.
- width: to set the width of the button.
- height: to set the height of the button.

## Entry Widgets

- An entry presents the user with a single line text field that they can use to type in a string value. These can be just about anything: their name, a city, a password, social security number, and so on.

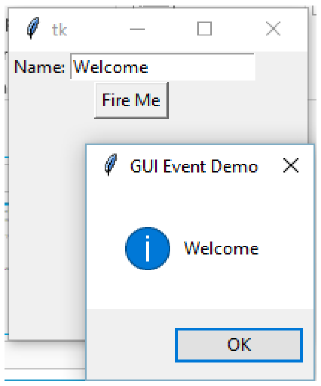
**Syntax:**

*name = ttk.Entry(parent, textvariable=username)*

## Entry Widgets

***Example:***

```
def hello():
   msg = messagebox.showinfo( "GUI Event Demo",t.get())
root = tk.Tk()
root.geometry("200x200")
l1=tk.Label(root,text="Name:")
l1.grid(row=0)
t=tk.Entry(root)
t.grid(row=0,column=1)
b = tk.Button(root, text='Fire Me',command=hello)
b.grid(row=1,columnspan=2);
root.mainloop()
```



## Canvas

- The Canvas is a rectangular area intended for drawing pictures or other complex layouts. You can place graphics, text, widgets or frames on a Canvas.
- It is used to draw pictures and other complex layout like graphics, text and widgets.

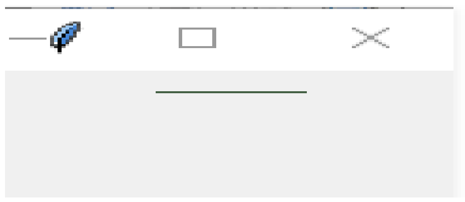
**Syntax:**

**w = Canvas(master, option=value)**

- master is the parameter used to represent the parent window.
- bd: to set the border width in pixels.
- bg: to set the normal background color.
- cursor: to set the cursor used in the canvas.
- highlightcolor: to set the color shown in the focus highlight.
- width: to set the width of the widget.
- height: to set the height of the widget.

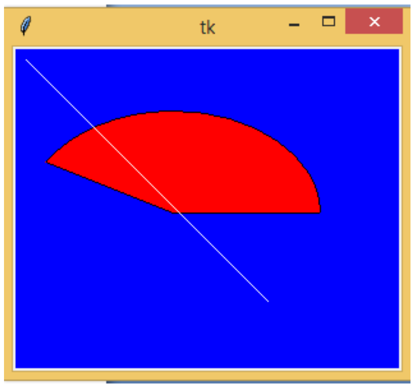
## Canvas

```
from tkinter import *
master = Tk()
w = Canvas(master, width=40, height=60)
w.pack()
canvas_height=20
canvas_width=200
y = int(canvas_height / 2)
w.create_line(0, y, canvas_width, y )
mainloop()
```



## Canvas

```
from tkinter import *

from tkinter import messagebox

top = Tk()

C = Canvas(top, bg = "blue", height = 250, width = 300)

coord = 10, 50, 240, 210

arc = C.create_arc(coord, start = 0, extent = 150, fill = "red")

line = C.create_line(10,10,200,200,fill = 'white')

C.pack()

top.mainloop()
```



## Checkbutton

- A checkbutton is like a regular button, except that not only can the user press it, which will invoke a command callback, but it also holds a binary value of some kind (i.e. a toggle). Checkbuttons are used all the time when a user is asked to choose between, e.g. two different values for an option.

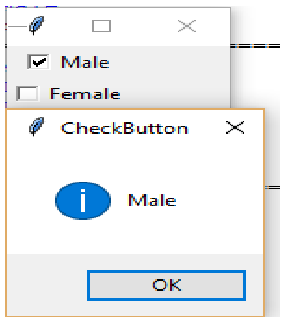
**Syntax**

**w = CheckButton(master, option=value)**

## Checkbutton

***Example:***

```
from tkinter import *
root= Tk()
root.title('Checkbutton Demo')
v1=IntVar()
v2=IntVar()
cb1=Checkbutton(root,text='Male', variable=v1,onvalue=1, offvalue=0, command=test)
cb1.grid(row=0)
cb2=Checkbutton(root,text='Female', variable=v2,onvalue=1, offvalue=0, command=test)
cb2.grid(row=1)
root.mainloop()
```



```
def test():
    if(v1.get()==1 ):
        v2.set(0)
        print("Male")
    if(v2.get()==1):
        v1.set(0)
        print("Female")
```

## Radiobutton

- A radiobutton lets you choose between one of a number of mutually exclusive choices; unlike a checkbutton, it is not limited to just two choices. Radiobuttons are always used together in a set and are a good option when the number of choices is fairly small

- **Syntax**

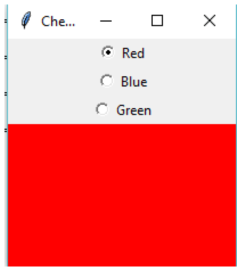
```
w = radioButton(master, option=value)
```

## radiobutton

***Example:***

```
root= Tk()
root.geometry("200x200")
radio=IntVar()
rb1=Radiobutton(root,text='Red', variable=radio,width=25,value=1, command=choice)
rb1.grid(row=0)
rb2=Radiobutton(root,text='Blue', variable=radio,width=25,value=2, command=choice)
rb2.grid(row=1)
rb3=Radiobutton(root,text='Green', variable=radio,width=25,value=3, command=choice)
rb3.grid(row=3)
root.mainloop()
```

```
def choice():
 if(radio.get()==1):
   root.configure(background='red')
 elif(radio.get()==2):
   root.configure(background='blue')
 elif(radio.get()==3):
   root.configure(background='green')
```



## Scale

- Scale widget is used to implement the graphical slider to the python application so that the user can slide through the range of values shown on the slider and select the one among them. We can control the minimum and maximum values along with the resolution of the scale. It provides an alternative to the Entry widget when the user is forced to select only one value from the given range of values.

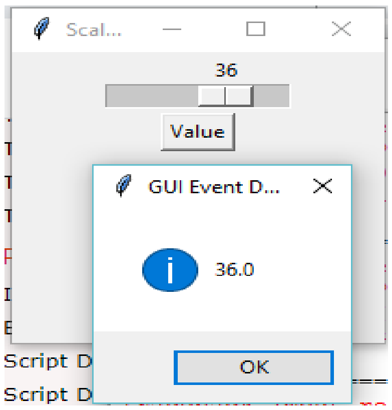
- **Syntax**

```
w = Scale(top, options)
```

## Scale

***Example:***

```
from tkinter import messagebox
root= Tk()
root.title('Scale Demo')
root.geometry("200x200")
def slide():
    msg = messagebox.showinfo( "GUI Event Demo",v.get())
v = DoubleVar()
scale = Scale( root, variable = v,  from_ = 1, to = 50, orient = HORIZONTAL)
scale.pack(anchor=CENTER)
btn = Button(root, text="Value", command=slide)
btn.pack(anchor=CENTER)
root.mainloop()
```



## Spinbox

- The Spinbox widget is an alternative to the Entry widget. It provides the range of values to the user, out of which, the user can select the one.

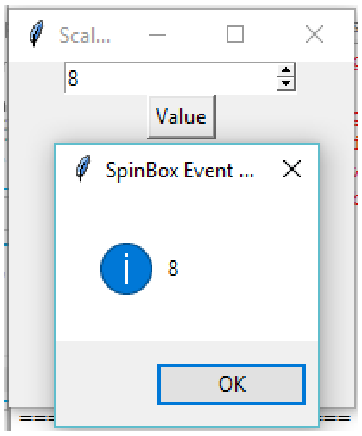
**Syntax**

**w = Spinbox(top, options)**

## Spinbox

***Example:***

```
from tkinter import *
from tkinter import messagebox

root= Tk()
root.title('Scale Demo')
root.geometry("200x200")

def slide():
    msg = messagebox.showinfo( "SpinBox Event Demo",spin.get())

spin = Spinbox(root, from_= 0, to = 25)
spin.pack(anchor=CENTER)
btn = Button(root, text="Value", command=slide)
btn.pack(anchor=CENTER)
root.mainloop()
```



## Menubutton

- Menubutton widget can be defined as the drop-down menu that is shown to the user all the time. It is used to provide the user a option to select the appropriate choice exist within the application.

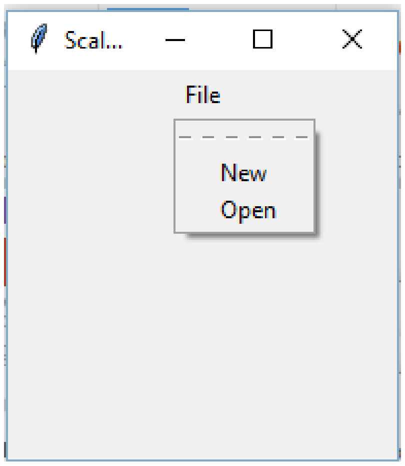
**Syntax**

**w = Menubutton(Top, options)**

## Menubutton

***Example:***

```
from tkinter import *
from tkinter import messagebox
root= Tk()
root.title('Scale Demo')
root.geometry("200x200")
menubutton = Menubutton(root, text = "File", relief = FLAT)
menubutton.grid()
menubutton.menu = Menu(menubutton)
menubutton["menu"]=menubutton.menu
menubutton.menu.add_checkbutton(label = "New",  variable=IntVar(),command=)
menubutton.menu.add_checkbutton(label = "Open", variable = IntVar())
menubutton.pack()
root.mainloop()
```



## Menubutton

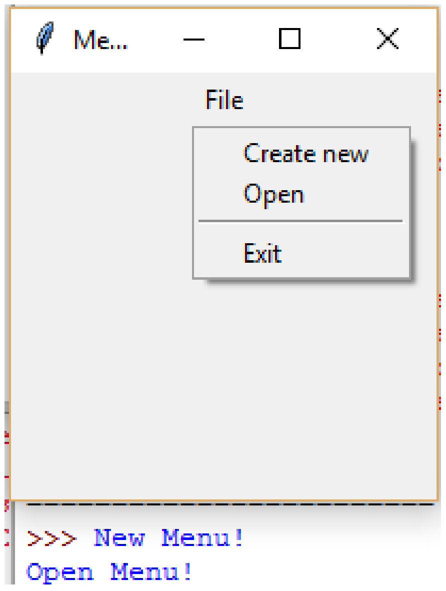
- Menubutton widget can be defined as the drop-down menu that is shown to the user all the time. It is used to provide the user a option to select the appropriate choice exist within the application.

**Syntax**

```
w = Menubutton(Top, options)
```

***Example:***

```
from tkinter import *
from tkinter import messagebox
root= Tk()
root.title('Menu Demo')
root.geometry("200x200")
def new():
  print("New Menu!")
def disp():
  print("Open Menu!")
menubutton = Menubutton(root, text="File")
menubutton.grid()
menubutton.menu = Menu(menubutton, tearoff = 0)
menubutton["menu"] = menubutton.menu
menubutton.menu.add_command(label="Create new",command=new)
menubutton.menu.add_command(label="Open",command=disp)
menubutton.menu.add_separator()
menubutton.menu.add_command(label="Exit",command=root.quit)
menubutton.pack()
```



```
>>> New Menu!
Open Menu!
```