Graphic User Interface

**import** **tkinter** **as** **tk**

**class** **App**(tk.Frame):

**def** \_\_init\_\_(self, master=**None**):

super().\_\_init\_\_(master)

self.pack()

*# create the application*

myapp = App()

*#*

*# here are method calls to the window manager class*

*#*

myapp.master.title("My Do-Nothing Application")

myapp.master.maxsize(1000, 400)

*# start the program*

myapp.mainloop()

**import** **tkinter**

widget = tkinter.Tk()

mask = tkinter.READABLE | tkinter.WRITABLE

widget.tk.createfilehandler(file, mask, callback)

...

widget.tk.deletefilehandler(file)

**import** **tkinter** **as** **tk**

**class** **App**(tk.Frame):

**def** \_\_init\_\_(self, master):

super().\_\_init\_\_(master)

self.pack()

self.entrythingy = tk.Entry()

self.entrythingy.pack()

*# Create the application variable.*

self.contents = tk.StringVar()

*# Set it to some value.*

self.contents.set("this is a variable")

*# Tell the entry widget to watch this variable.*

self.entrythingy["textvariable"] = self.contents

*# Define a callback for when the user hits return.*

*# It prints the current value of the variable.*

self.entrythingy.bind('<Key-Return>',

self.print\_contents)

**def** print\_contents(self, event):

print("Hi. The current entry content is:",

self.contents.get())

root = tk.Tk()

myapp = App(root)

myapp.mainloop()

**import** **gtk**

**def** create\_window():

window = gtk.Window()

window.set\_default\_size(200, 200)

window.connect("destroy", gtk.main\_quit)

label = gtk.Label("Hello World")

window.add(label)

label.show()

window.show()

create\_window()

gtk.main()

**from** **kivy.app** **import** App

**from** **kivy.uix.button** **import** Button

**class** **TestApp**(App):

**def** build(self):

**return** Button(text="Hello World")

TestApp().run()

**<LoadDialog>**:

**BoxLayout**:

**size**: root.size

**pos**: root.pos

**orientation**: "vertical"

**FileChooserListView**:

**id**: filechooser

**BoxLayout**:

**size\_hint\_y**: None

**height**: 30

**Button**:

**text**: "Cancel"

**on\_release**: root.cancel()

**Button**:

**text**: "Load"

**on\_release**: root.load(filechooser.path, filechooser.selection)

**import** **wx**

**class** **WxButton**(wx.Frame):

**def** \_\_init\_\_(self, \*args, \*\*kw):

super(WxButton, self).\_\_init\_\_(\*args, \*\*kw)

self.InitUI()

**def** InitUI(self):

pnl = wx.Panel(self)

closeButton = wx.Button(pnl, label='Close Me', pos=(20, 20))

closeButton.Bind(wx.EVT\_BUTTON, self.OnClose)

self.SetSize((350, 250))

self.SetTitle('Close Button')

self.Centre()

**def** OnClose(self, e):

self.Close(**True**)

**def** main():

app = wx.App()

ex = WxButton(**None**)

ex.Show()

app.MainLoop()

**if** \_\_name\_\_ == "\_\_main\_\_":

main()

**import** **wx**

app = wx.App(**False**) *# Create a new app, don't redirect stdout/stderr to a window.*

frame = wx.Frame(**None**, title="Hello World") *# A Frame is a top-level window.*

frame.Show(**True**) *# Show the frame.*

app.MainLoop()

import javax.swing.\*;

class gui{

public static void main(String args[]){

JFrame frame = new JFrame("My First GUI");

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setSize(300,300);

JButton button = new JButton("Press");

frame.getContentPane().add(button); // Adds Button to content pane of frame

frame.setVisible(true);

}

}

import javax.swing.\*;

class gui{

public static void main(String args[]){

JFrame frame = new JFrame("My First GUI");

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setSize(300,300);

JButton button1 = new JButton("Press");

frame.getContentPane().add(button1);

frame.setVisible(true);

}

}

import javax.swing.\*;

class gui{

public static void main(String args[]){

JFrame frame = new JFrame("My First GUI");

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setSize(300,300);

JButton button1 = new JButton("Button 1");

JButton button2 = new JButton("Button 2");

frame.getContentPane().add(button1);

frame.getContentPane().add(button2);

frame.setVisible(true);

}

}

//Usually you will require both swing and awt packages

// even if you are working with just swings.

import javax.swing.\*;

import java.awt.\*;

class gui {

public static void main(String args[]) {

//Creating the Frame

JFrame frame = new JFrame("Chat Frame");

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setSize(400, 400);

//Creating the MenuBar and adding components

JMenuBar mb = new JMenuBar();

JMenu m1 = new JMenu("FILE");

JMenu m2 = new JMenu("Help");

mb.add(m1);

mb.add(m2);

JMenuItem m11 = new JMenuItem("Open");

JMenuItem m22 = new JMenuItem("Save as");

m1.add(m11);

m1.add(m22);

//Creating the panel at bottom and adding components

JPanel panel = new JPanel(); // the panel is not visible in output

JLabel label = new JLabel("Enter Text");

JTextField tf = new JTextField(10); // accepts upto 10 characters

JButton send = new JButton("Send");

JButton reset = new JButton("Reset");

panel.add(label); // Components Added using Flow Layout

panel.add(tf);

panel.add(send);

panel.add(reset);

// Text Area at the Center

JTextArea ta = new JTextArea();

//Adding Components to the frame.

frame.getContentPane().add(BorderLayout.SOUTH, panel);

frame.getContentPane().add(BorderLayout.NORTH, mb);

frame.getContentPane().add(BorderLayout.CENTER, ta);

frame.setVisible(true);

}

}

**import** **java.awt.\***;

**import** **java.awt.event.WindowAdapter**;

**import** **java.awt.event.WindowEvent**;

**public** **class** **MyApp** {

**public** **static** void main(String[] args) {

Frame frame = **new** Frame("Application");

frame.add(**new** Label("Hello!"));

frame.setSize(500, 500);

frame.setLocationRelativeTo(**null**); *// Centers the window*

frame.addWindowListener(**new** WindowAdapter() {

@Override

**public** void windowClosing(WindowEvent e) {

frame.dispose(); *// Releases native screen resources*

}

});

frame.setVisible(**true**);

}

}

import java.awt.\*;

    class FrameButton{

        FrameButton (){

            Frame f=new Frame();

            Button b=new Button("CLICK\_ME");

            b.setBounds(30,50,80,30);

            f.add(b);

            f.setSize(300,300);

            f.setLayout(null);

        f.setVisible(true);

    }

    public static void main(String args[]){

        FrameButton f=new FrameButton ();

    }

}

import java.awt.\*;

class AWTButton extends Frame{

    AWTButton (){

    Button b=new Button("AWTButton");

    b.setBounds(30,100,80,30);// setting button position

    add(b);//adding button into frame

    setSize(300,300);//frame size 300 width and 300 height

    setLayout(null);//no layout manager

    setVisible(true);//now frame will be visible, by default not visible

}

    public static void main(String args[]){

        AWTButton f=new AWTButton ();

    }

}

// Include gtk

#include <gtk/gtk.h>

**static** **void** **on\_activate** (GtkApplication \*app) {

// Create a new window

GtkWidget \*window = gtk\_application\_window\_new (app);

// Create a new button

GtkWidget \*button = gtk\_button\_new\_with\_label ("Hello, World!");

// When the button is clicked, close the window passed as an argument

g\_signal\_connect\_swapped (button, "clicked", G\_CALLBACK (gtk\_window\_close), window);

gtk\_window\_set\_child (GTK\_WINDOW (window), button);

gtk\_window\_present (GTK\_WINDOW (window));

}

**int** **main** (**int** argc, **char** \*argv[]) {

// Create a new application

GtkApplication \*app = gtk\_application\_new ("com.example.GtkApplication",

G\_APPLICATION\_FLAGS\_NONE);

g\_signal\_connect (app, "activate", G\_CALLBACK (on\_activate), NULL);

**return** g\_application\_run (G\_APPLICATION (app), argc, argv);

}

imports.gi.versions['Gtk'] = '4.0';

**const** Gtk = imports.gi.Gtk;

// Create a new application

**let** app = **new** Gtk.Application({ application\_id: 'com.example.GtkApplication' });

// When the application is launched…

app.connect('activate', () => {

// … create a new window …

**let** win = **new** Gtk.ApplicationWindow({ application: app });

// … with a button in it …

**let** btn = **new** Gtk.Button({ label: 'Hello, World!' });

// … which closes the window when clicked

btn.connect('clicked', () => { win.close(); });

win.set\_child(btn);

win.present();

});

#!/usr/bin/env perl

**use** strict; **use** warnings; **use** utf8;

**use** **Glib::**IO;

**use** Gtk4;

# Create a new application

**my** $app =

**Gtk3::**Application->**new**('com.example.Gtk3Application', 'G\_APPLICATION\_FLAGS\_NONE');

# When the application is launched…

$app->signal\_connect(

activate => **sub** {

**my** $app = shift;

# … create a new window …

**my** $win = **Gtk3::**ApplicationWindow->**new**($app);

# … with a button in it …

**my** $btn = **Gtk3::**Button->**new**('Hello World!');

# … which closes the window when clicked

$btn->signal\_connect(clicked => **sub** { $win->close(); });

$win->set\_child($btn);

$win->present();

}

);

# Run the application

$app->run(\@ARGV);

# Load Gtk

**import** **gi**

gi.require\_version('Gtk', '4.0')

**from** **gi.repository** **import** Gtk

# When the application is launched…

**def** **on\_activate**(app):

# … create a new window…

win = Gtk.ApplicationWindow(application=app)

# … with a button in it…

btn = Gtk.Button(label='Hello, World!')

# … which closes the window when clicked

btn.connect('clicked', **lambda** x: win.close())

win.set\_child(btn)

win.present()

# Create a new application

app = Gtk.Application(application\_id='com.example.GtkApplication')

app.connect('activate', on\_activate)

# Run the application

app.run(None)

**use** **glib**::clone;

// glib and other dependencies are re-exported by the gtk crate

**use** **gtk**::glib;

**use** **gtk**::**prelude**::\*;

// When the application is launched…

**fn** **on\_activate**(application: &**gtk**::Application) {

// … create a new window …

**let** window = **gtk**::**ApplicationWindow**::**new**(application);

// … with a button in it …

**let** button = **gtk**::**Button**::**with\_label**("Hello World!");

// … which closes the window when clicked

button**.connect\_clicked**(**clone!**(@weak window **=>** **move** |**\_**| window**.close**()));

window**.set\_child**(**Some**(&button));

window**.present**();

}

**fn** **main**() {

// Create a new application with the builder pattern

**let** app = **gtk**::**Application**::**builder**()

**.application\_id**("com.github.gtk-rs.examples.basic")

**.build**();

app**.connect\_activate**(on\_activate);

// Run the application

app**.run**();

}

**int** **main** (**string**[] argv) {

// Create a new application

**var** app = **new** Gtk.**Application** ("com.example.GtkApplication",

GLib.ApplicationFlags.FLAGS\_NONE);

app.activate.**connect** (() => {

// Create a new window

**var** window = **new** Gtk.**ApplicationWindow** (app);

// Create a new button

**var** button = **new** Gtk.Button.**with\_label** ("Hello, World!");

// When the button is clicked, close the window

button.clicked.**connect** (() => {

window.**close** ();

});

window.**set\_child** (button);

window.**present** ();

});

**return** app.**run** (argv);

}

import Cocoa

class ClearWindow : NSWindow {

override init(contentRect: NSRect, styleMask style: NSWindow.StyleMask, backing backingStoreType: NSWindow.BackingStoreType, defer flag: Bool) {

super.init(contentRect: contentRect, styleMask: StyleMask.borderless, backing: backingStoreType, defer: flag)

level = NSWindow.Level.statusBar

backgroundColor = NSColor.blue

}

override func mouseDown(with event: NSEvent) {

print("Mouse down: \(event.locationInWindow)")

}

override func mouseDragged(with event: NSEvent) {

print("Mouse dragged: \(event.locationInWindow)")

}

override func mouseUp(with event: NSEvent) {

print("Mouse up: \(event.locationInWindow)")

}

}

import Cocoa

class ViewController: NSViewController {

override func viewDidLoad() {

super.viewDidLoad()

view.frame = CGRect(origin: CGPoint(), size: NSScreen.main!.visibleFrame.size)

}

func startDrawing(at point: NSPoint) {

}

func continueDrawing(at point: NSPoint) {

}

func endDrawing(at point: NSPoint) {

}

}

let lineWeight: CGFloat = 10

let strokeColor: NSColor = .red

var currentPath: NSBezierPath?

var currentShape: CAShapeLayer?

func startDrawing(at point: NSPoint) {

currentPath = NSBezierPath()

currentShape = CAShapeLayer()

currentShape?.lineWidth = lineWeight

currentShape?.strokeColor = strokeColor.cgColor

currentShape?.fillColor = NSColor.clear.cgColor

currentShape?.lineJoin = kCALineJoinRound

currentShape?.lineCap = kCALineCapRound

currentPath?.move(to: point)

currentPath?.line(to: point)

currentShape?.path = currentPath?.cgPath

view.layer?.addSublayer(currentShape!)

}

func continueDrawing(at point: NSPoint) {

currentPath?.line(to: point)

if let shape = currentShape {

shape.path = currentPath?.cgPath

}

}

func endDrawing(at point: NSPoint) {

currentPath?.line(to: point)

if let shape = currentShape {

shape.path = currentPath?.cgPath

}

currentPath = nil

currentShape = nil

}

import Cocoa

extension NSBezierPath {

public var cgPath: CGPath {

let path = CGMutablePath()

var points = [CGPoint](repeating: .zero, count: 3)

for i in 0 ..< self.elementCount {

let type = self.element(at: i, associatedPoints: &points)

switch type {

case .moveToBezierPathElement:

path.move(to: points[0])

case .lineToBezierPathElement:

path.addLine(to: points[0])

case .curveToBezierPathElement:

path.addCurve(to: points[2], control1: points[0], control2: points[1])

case .closePathBezierPathElement:

path.closeSubpath()

}

}

return path

}

}

override func mouseDown(with event: NSEvent) {

(contentViewController as? ViewController)?.startDrawing(at: event.locationInWindow)

}

override func mouseDragged(with event: NSEvent) {

(contentViewController as? ViewController)?.continueDrawing(at: event.locationInWindow)

}

override func mouseUp(with event: NSEvent) {

(contentViewController as? ViewController)?.endDrawing(at: event.locationInWindow)

}

@IBOutlet weak var clearButton: NSMenuItem!

@IBOutlet weak var toggleButton: NSMenuItem!

@IBOutlet var optionsMenu: NSMenu!

@IBAction func clearButtonClicked(\_ sender: Any) {

}

@IBAction func toggleButtonClicked(\_ sender: Any) {

}

private let offText = "Disable Drawing"

private let onText = "Enable Drawing"

view.window!.ignoresMouseEvents = !view.window!.ignoresMouseEvents

toggleButton.title = view.window!.ignoresMouseEvents ? onText : offText

let statusItem = NSStatusBar.system.statusItem(withLength: NSStatusItem.variableLength)

override func awakeFromNib() {

statusItem.menu = optionsMenu

let icon = NSImage(named: NSImage.Name(rawValue: "pencil"))

icon?.isTemplate = true // best for dark mode

statusItem.image = icon

toggleButton.title = offText

}

backgroundColor = NSColor(calibratedRed: 1, green: 1, blue: 1, alpha: 0.001)

let lineWeight: CGFloat = 10

let strokeColor: NSColor = .red

var currentPath: NSBezierPath?

var currentShape: CAShapeLayer?

func startDrawing(at point: NSPoint) {

currentPath = NSBezierPath()

currentShape = CAShapeLayer()

currentShape?.lineWidth = lineWeight

currentShape?.strokeColor = strokeColor.cgColor

currentShape?.fillColor = NSColor.clear.cgColor

currentShape?.lineJoin = kCALineJoinRound

currentShape?.lineCap = kCALineCapRound

currentPath?.move(to: point)

currentPath?.line(to: point)

currentShape?.path = currentPath?.cgPath

view.layer?.addSublayer(currentShape!)

}

func continueDrawing(at point: NSPoint) {

currentPath?.line(to: point)

if let shape = currentShape {

shape.path = currentPath?.cgPath

}

}

func endDrawing(at point: NSPoint) {

currentPath?.line(to: point)

if let shape = currentShape {

shape.path = currentPath?.cgPath

}

currentPath = nil

currentShape = nil

}

import Cocoa

extension NSBezierPath {

public var cgPath: CGPath {

let path = CGMutablePath()

var points = [CGPoint](repeating: .zero, count: 3)

for i in 0 ..< self.elementCount {

let type = self.element(at: i, associatedPoints: &points)

switch type {

case .moveToBezierPathElement:

path.move(to: points[0])

case .lineToBezierPathElement:

path.addLine(to: points[0])

case .curveToBezierPathElement:

path.addCurve(to: points[2], control1: points[0], control2: points[1])

case .closePathBezierPathElement:

path.closeSubpath()

}

}

return path

}

}

override func mouseDown(with event: NSEvent) {

(contentViewController as? ViewController)?.startDrawing(at: event.locationInWindow)

}

override func mouseDragged(with event: NSEvent) {

(contentViewController as? ViewController)?.continueDrawing(at: event.locationInWindow)

}

override func mouseUp(with event: NSEvent) {

(contentViewController as? ViewController)?.endDrawing(at: event.locationInWindow)

}

@IBOutlet weak var clearButton: NSMenuItem!

@IBOutlet weak var toggleButton: NSMenuItem!

@IBOutlet var optionsMenu: NSMenu!

@IBAction func clearButtonClicked(\_ sender: Any) {

}

@IBAction func toggleButtonClicked(\_ sender: Any) {

}

private let offText = "Disable Drawing"

private let onText = "Enable Drawing"

view.window!.ignoresMouseEvents = !view.window!.ignoresMouseEvents

toggleButton.title = view.window!.ignoresMouseEvents ? onText : offText

let statusItem = NSStatusBar.system.statusItem(withLength: NSStatusItem.variableLength)

override func awakeFromNib() {

statusItem.menu = optionsMenu

let icon = NSImage(named: NSImage.Name(rawValue: "pencil"))

icon?.isTemplate = true // best for dark mode

statusItem.image = icon

toggleButton.title = offText

}

backgroundColor = NSColor(calibratedRed: 1, green: 1, blue: 1, alpha: 0.001)

@main

struct MyApp: App {

var body: some Scene {

WindowGroup {

HomePage()

}

}

}

struct HomePage: View {

var body: some View {

Text("Hello, World!")

}

}

void main() {

runApp(const MyApp());

}

class MyApp extends StatelessWidget {

const MyApp({super.key});

@override

Widget build(BuildContext context) {

// Returns a CupertinoApp that, by default,

// has the look and feel of an iOS app.

return const CupertinoApp(

home: HomePage(),

);

}

}

class HomePage extends StatelessWidget {

const HomePage({super.key});

@override

Widget build(BuildContext context) {

return const Scaffold(

body: Center(

child: Text(

'Hello, World!',

),

),

);

}

}

Button("Do something") {

// this closure gets called when your

// button is tapped

}

CupertinoButton(

onPressed: () {

// This closure is called when your button is tapped.

},

child: const Text('Do something'),

)

HStack {

Image(systemName: "globe")

Text("Hello, world!")

}

Row(

mainAxisAlignment: MainAxisAlignment.center,

children: [

Icon(CupertinoIcons.globe),

Text('Hello, world!'),

],

),

VStack {

Image(systemName: "globe")

Text("Hello, world!")

}

Column(

mainAxisAlignment: MainAxisAlignment.center,

children: [

Icon(CupertinoIcons.globe),

Text('Hello, world!'),

],

),

struct Person: Identifiable {

var name: String

}

var persons = [

Person(name: "Person 1"),

Person(name: "Person 2"),

Person(name: "Person 3"),

]

struct ListWithPersons: View {

let persons: [Person]

var body: some View {

List {

ForEach(persons) { person in

Text(person.name)

}

}

}

}

class Person {

String name;

Person(this.name);

}

var items = [

Person('Person 1'),

Person('Person 2'),

Person('Person 3'),

];

class HomePage extends StatelessWidget {

const HomePage({super.key});

@override

Widget build(BuildContext context) {

return Scaffold(

body: ListView.builder(

itemCount: items.length,

itemBuilder: (context, index) {

return ListTile(

title: Text(items[index].name),

);

},

),

);

}

}

Grid {

GridRow {

Text("Row 1")

Image(systemName: "square.and.arrow.down")

Image(systemName: "square.and.arrow.up")

}

GridRow {

Text("Row 2")

Image(systemName: "square.and.arrow.down")

Image(systemName: "square.and.arrow.up")

}

}

const widgets = [

Text('Row 1'),

Icon(CupertinoIcons.arrow\_down\_square),

Icon(CupertinoIcons.arrow\_up\_square),

Text('Row 2'),

Icon(CupertinoIcons.arrow\_down\_square),

Icon(CupertinoIcons.arrow\_up\_square),

];

class HomePage extends StatelessWidget {

const HomePage({super.key});

@override

Widget build(BuildContext context) {

return Scaffold(

body: GridView.builder(

gridDelegate: const SliverGridDelegateWithFixedCrossAxisCount(

crossAxisCount: 3,

mainAxisExtent: 40,

),

itemCount: widgets.length,

itemBuilder: (context, index) => widgets[index],

),

);

}

}

ScrollView {

VStack(alignment: .leading) {

ForEach(persons) { person in

PersonView(person: person)

}

}

}

SingleChildScrollView(

child: Column(

children: mockPersons

.map(

(person) => PersonView(

person: person,

),

)

.toList(),

),

),

struct ContentView: View {

@State private var counter = 0;

var body: some View {

VStack{

Button("+") { counter+=1 }

Text(String(counter))

}

}}

class MyHomePage extends StatefulWidget {

const MyHomePage({super.key});

@override

State<MyHomePage> createState() => \_MyHomePageState();

}

class \_MyHomePageState extends State<MyHomePage> {

int \_counter = 0;

@override

Widget build(BuildContext context) {

return Scaffold(

body: Center(

child: Column(

mainAxisAlignment: MainAxisAlignment.center,

children: [

Text('$\_counter'),

TextButton(

onPressed: () => setState(() {

\_counter++;

}),

child: const Text('+'),

),

],

),

),

);

}

}

Button(“Tap me!”){

angle += 45

}

.rotationEffect(.degrees(angle))

.animation(.easeIn(duration: 1))

AnimatedRotation(

duration: const Duration(seconds: 1),

turns: turns,

curve: Curves.easeIn,

child: TextButton(

onPressed: () {

setState(() {

turns += .125;

});

},

child: const Text('Tap me!')),

),

CustomPaint(

painter: SignaturePainter(\_points),

size: Size.infinite,

),

class SignaturePainter extends CustomPainter {

SignaturePainter(this.points);

final List<Offset?> points;

@override

void paint(Canvas canvas, Size size) {

final Paint paint = Paint()

..color = Colors.black

..strokeCap = StrokeCap.round

..strokeWidth = 5;

for (int i = 0; i < points.length - 1; i++) {

if (points[i] != null && points[i + 1] != null) {

canvas.drawLine(points[i]!, points[i + 1]!, paint);

}

}

}

@override

bool shouldRepaint(SignaturePainter oldDelegate) =>

oldDelegate.points != points;

}

NavigationStack(path: $path) {

List {

ForEach(persons) { person in

NavigationLink(

person.name,

value: person

)

}

}

.navigationDestination(for: Person.self) { person in

PersonView(person: person)

}

}

// Defines the route name as a constant

// so that it's reusable.

const detailsPageRouteName = '/details';

class App extends StatelessWidget {

const App({

super.key,

});

@override

Widget build(BuildContext context) {

return CupertinoApp(

home: const HomePage(),

// The [routes] property defines the available named routes

// and the widgets to build when navigating to those routes.

routes: {

detailsPageRouteName: (context) => const DetailsPage(),

},

);

}

}

ListView.builder(

itemCount: mockPersons.length,

itemBuilder: (context, index) {

final person = mockPersons.elementAt(index);

final age = '${person.age} years old';

return ListTile(

title: Text(person.name),

subtitle: Text(age),

trailing: const Icon(

Icons.arrow\_forward\_ios,

),

onTap: () {

// When a [ListTile] that represents a person is

// tapped, push the detailsPageRouteName route

// to the Navigator and pass the person's instance

// to the route.

Navigator.of(context).pushNamed(

detailsPageRouteName,

arguments: person,

);

},

);

},

),

class DetailsPage extends StatelessWidget {

const DetailsPage({super.key});

@override

Widget build(BuildContext context) {

// Read the person instance from the arguments.

final Person person = ModalRoute.of(

context,

)?.settings.arguments as Person;

// Extract the age.

final age = '${person.age} years old';

return Scaffold(

// Display name and age.

body: Column(children: [Text(person.name), Text(age)]),

);

}

}

Button("Pop back") {

dismiss()

}

TextButton(

onPressed: () {

// This code allows the

// view to pop back to its presenter.

Navigator.of(context).pop();

},

child: const Text('Pop back'),

),

@Environment(\.openURL) private var openUrl

// View code goes here

Button("Open website") {

openUrl(

URL(

string: "https://google.com"

)!

)

}

CupertinoButton(

onPressed: () async {

await launchUrl(

Uri.parse('https://google.com'),

);

},

child: const Text(

'Open website',

),

),

CupertinoApp(

theme: CupertinoThemeData(

brightness: Brightness.dark,

),

home: HomePage(),

);

Text("Hello, world!")

.font(.system(size: 30, weight: .heavy))

.foregroundColor(.yellow)

Text(

'Hello, world!',

style: TextStyle(

fontSize: 30,

fontWeight: FontWeight.bold,

color: CupertinoColors.systemYellow,

),

),

Button("Do something") {

// do something when button is tapped

}

.font(.system(size: 30, weight: .bold))

.background(Color.yellow)

.foregroundColor(Color.blue)

}

child: CupertinoButton(

color: CupertinoColors.systemYellow,

onPressed: () {},

padding: const EdgeInsets.all(16),

child: const Text(

'Do something',

style: TextStyle(

color: CupertinoColors.systemBlue,

fontSize: 30,

fontWeight: FontWeight.bold,

),

),

),

Text("Hello")

.font(

Font.custom(

"BungeeSpice-Regular",

size: 40

)

)

flutter:

fonts:

- family: BungeeSpice

fonts:

- asset: fonts/BungeeSpice-Regular.ttf

Text(

'Cupertino',

style: TextStyle(

fontSize: 40,

fontFamily: 'BungeeSpice',

),

)

flutter:

assets:

- images/Blueberries.jpg

<?php

**function** pressed()

{

**echo** "Hello again - The button was pressed!**\n**";

}

$window = **new** GtkWindow();

$button = **new** GtkButton('Click');

$button1 = **new** GtkButton('Click');

$window->set\_title('Hello World!');

$window->connect\_simple('destroy', **array**('Gtk', 'main\_quit'));

$button->connect\_simple('clicked', 'pressed');

$button1->connect\_simple('clicked', 'pressed');

$window->add($button);

$window->show\_all();

Gtk::main();

class HelloMessage extends React.Component {

render() {

return <div>Hello {this.props.name}</div>;

}

}

root.render(<HelloMessage name="Taylor" />);

class Timer extends React.Component {

constructor(props) {

super(props);

this.state = { seconds: 0 };

}

tick() {

this.setState(state => ({

seconds: state.seconds + 1

}));

}

componentDidMount() {

this.interval = setInterval(() => this.tick(), 1000);

}

componentWillUnmount() {

clearInterval(this.interval);

}

render() {

return (

<div>

Seconds: {this.state.seconds}

</div>

);

}

}

root.render(<Timer />);

class TodoApp extends React.Component {

constructor(props) {

super(props);

this.state = { items: [], text: '' };

this.handleChange = this.handleChange.bind(this);

this.handleSubmit = this.handleSubmit.bind(this);

}

render() {

return (

<div>

<h3>TODO</h3>

<TodoList items={this.state.items} />

<form onSubmit={this.handleSubmit}>

<label htmlFor="new-todo">

What needs to be done?

</label>

<input

id="new-todo"

onChange={this.handleChange}

value={this.state.text}

/>

<button>

Add #{this.state.items.length + 1}

</button>

</form>

</div>

);

}

handleChange(e) {

this.setState({ text: e.target.value });

}

handleSubmit(e) {

e.preventDefault();

if (this.state.text.length === 0) {

return;

}

const newItem = {

text: this.state.text,

id: Date.now()

};

this.setState(state => ({

items: state.items.concat(newItem),

text: ''

}));

}

}

class TodoList extends React.Component {

render() {

return (

<ul>

{this.props.items.map(item => (

<li key={item.id}>{item.text}</li>

))}

</ul>

);

}

}

root.render(<TodoApp />);

class MarkdownEditor extends React.Component {

constructor(props) {

super(props);

this.md = new Remarkable();

this.handleChange = this.handleChange.bind(this);

this.state = { value: 'Hello, \*\*world\*\*!' };

}

handleChange(e) {

this.setState({ value: e.target.value });

}

getRawMarkup() {

return { \_\_html: this.md.render(this.state.value) };

}

<**template**>

<**div** id="tuto">

<**button-clicked** v-bind:initial-count="0"></**button-clicked**>

</**div**>

</**template**>

<**script**>

Vue.component('button-clicked', {

props: ['initialCount'],

data: () => ({

count: 0,

}),

template: '<button v-on:click="onClick">Clicked {{ count }} times</button>',

computed: {

countTimesTwo() {

**return** **this**.count \* 2;

}

},

watch: {

count(newValue, oldValue) {

console.log(`The value of count is changed from **${**oldValue**}** to **${**newValue**}**.`);

}

},

methods: {

onClick() {

**this**.count += 1;

}

},

mounted() {

**this**.count = **this**.initialCount;

}

});

**new** Vue({

el: '#tuto',

});

</**script**>

<**div** id="app">

<**router-view**></**router-view**>

</**div**>

...

<**script**>

...

**const** User = {

template: '<div>User {{ $route.params.id }}</div>'

};

**const** router = **new** VueRouter({

routes: [

{ path: '/user/:id', component: User }

]

});

...

</**script**>

<**div** id="app">

<**div**>

<**div**>User 1</**div**>

</**div**>

</**div**>

// Variable overrides first

$primary: #900;

$enable-shadows: true;

$prefix: "mo-";

// Then import Bootstrap

@import "../node\_modules/bootstrap/scss/bootstrap";

// Functions first

@import "../node\_modules/bootstrap/scss/functions";

// Variable overrides second

$primary: #900;

$enable-shadows: true;

$prefix: "mo-";

// Required Bootstrap imports

@import "../node\_modules/bootstrap/scss/variables";

@import "../node\_modules/bootstrap/scss/variables-dark";

@import "../node\_modules/bootstrap/scss/maps";

@import "../node\_modules/bootstrap/scss/mixins";

@import "../node\_modules/bootstrap/scss/root";

// Optional components

@import "../node\_modules/bootstrap/scss/utilities";

@import "../node\_modules/bootstrap/scss/reboot";

@import "../node\_modules/bootstrap/scss/containers";

@import "../node\_modules/bootstrap/scss/grid";

@import "../node\_modules/bootstrap/scss/helpers";

@import "../node\_modules/bootstrap/scss/utilities/api";

.component {

color: var(--bs-gray-800);

background-color: var(--bs-gray-100);

border: 1px solid var(--bs-gray-200);

border-radius: .25rem;

}

.component-header {

color: var(--bs-purple);

}

body {

--bs-body-font-family: var(--bs-font-monospace);

--bs-body-line-height: 1.4;

--bs-body-bg: var(--bs-gray-100);

}

.table {

--bs-table-color: var(--bs-gray-600);

--bs-table-bg: var(--bs-gray-100);

--bs-table-border-color: transparent;

}

<ul class="nav nav-pills nav-fill gap-2 p-1 small bg-primary rounded-5 shadow-sm" id="pillNav2" role="tablist" style="--bs-nav-link-color: var(--bs-white); --bs-nav-pills-link-active-color: var(--bs-primary); --bs-nav-pills-link-active-bg: var(--bs-white);">

<li class="nav-item" role="presentation">

<button class="nav-link active rounded-5" id="home-tab2" data-bs-toggle="tab" type="button" role="tab" aria-selected="true">Home</button>

</li>

<li class="nav-item" role="presentation">

<button class="nav-link rounded-5" id="profile-tab2" data-bs-toggle="tab" type="button" role="tab" aria-selected="false">Profile</button>

</li>

<li class="nav-item" role="presentation">

<button class="nav-link rounded-5" id="contact-tab2" data-bs-toggle="tab" type="button" role="tab" aria-selected="false">Contact</button>

</li>

@import "bootstrap/scss/bootstrap";

$utilities: map-merge(

$utilities,

(

"cursor": (

property: cursor,

class: cursor,

responsive: true,

values: auto pointer grab,

)

)

);

<div class="dropdown">

<button class="btn btn-primary dropdown-toggle" type="button" data-bs-toggle="dropdown" aria-expanded="false">

Dropdown

</button>

<ul class="dropdown-menu">

<li><button class="dropdown-item" type="button">Dropdown item</button></li>

<li><button class="dropdown-item" type="button">Dropdown item</button></li>

<li><button class="dropdown-item" type="button">Dropdown item</button></li>

</ul>

**Hello World GUI:**

import tkinter as tk

def say\_hello():

label.config(text="Hello, " + entry.get())

root = tk.Tk()

root.title("Hello World GUI")

label = tk.Label(root, text="Enter your name:")

label.pack()

entry = tk.Entry(root)

entry.pack()

button = tk.Button(root, text="Say Hello", command=say\_hello)

button.pack()

root.mainloop()

**Calculator GUI:**

import tkinter as tk

def button\_click(number):

current = entry.get()

entry.delete(0, tk.END)

entry.insert(0, current + str(number))

def clear():

entry.delete(0, tk.END)

def calculate():

result = eval(entry.get())

entry.delete(0, tk.END)

entry.insert(0, result)

root = tk.Tk()

root.title("Calculator")

entry = tk.Entry(root, width=20)

entry.grid(row=0, column=0, columnspan=4)

buttons = [

'7', '8', '9', '/',

'4', '5', '6', '\*',

'1', '2', '3', '-',

'0', '.', '=', '+'

]

row\_val = 1

col\_val = 0

for button\_text in buttons:

tk.Button(root, text=button\_text, padx=20, pady=20, command=lambda button\_text=button\_text: button\_click(button\_text) if button\_text != '=' else calculate()).grid(row=row\_val, column=col\_val)

col\_val += 1

if col\_val > 3:

col\_val = 0

row\_val += 1

tk.Button(root, text='C', padx=20, pady=20, command=clear).grid(row=row\_val, column=col\_val)

root.mainloop()

**Image Viewer GUI:**

import tkinter as tk

from PIL import Image, ImageTk

def show\_image():

file\_path = file\_entry.get()

image = Image.open(file\_path)

photo = ImageTk.PhotoImage(image)

image\_label.config(image=photo)

image\_label.photo = photo

root = tk.Tk()

root.title("Image Viewer")

file\_label = tk.Label(root, text="Enter file path:")

file\_label.pack()

file\_entry = tk.Entry(root)

file\_entry.pack()

show\_button = tk.Button(root, text="Show Image", command=show\_image)

show\_button.pack()

image\_label = tk.Label(root)

image\_label.pack()

root.mainloop()

**Simple Calculator GUI:**

import tkinter as tk

def calculate():

result.set(eval(entry.get()))

root = tk.Tk()

root.title("Simple Calculator")

entry = tk.Entry(root)

entry.pack()

result = tk.StringVar()

label = tk.Label(root, textvariable=result)

label.pack()

button = tk.Button(root, text="Calculate", command=calculate)

button.pack()

root.mainloop()

**File Dialog GUI:**

import tkinter as tk

from tkinter import filedialog

def open\_file():

file\_path = filedialog.askopenfilename()

result.set(file\_path)

root = tk.Tk()

result = tk.StringVar()

button = tk.Button(root, text="Open File", command=open\_file)

button.pack()

label = tk.Label(root, textvariable=result)

label.pack()

root.mainloop()

**Simple GUI with Multiple Widgets:**

import tkinter as tk

def submit():

result.set(f"Hello, {entry.get()}!")

root = tk.Tk()

root.title("Simple GUI")

label = tk.Label(root, text="Enter your name:")

label.pack()

entry = tk.Entry(root)

entry.pack()

result = tk.StringVar()

result\_label = tk.Label(root, textvariable=result)

result\_label.pack()

button = tk.Button(root, text="Submit", command=submit)

button.pack()

root.mainloop()

**Hello World Window:**

import tkinter as tk

# Create a main window

root = tk.Tk()

# Create a label widget

label = tk.Label(root, text="Hello, World!")

# Pack the label into the window

label.pack()

# Start the main event loop

root.mainloop()

**Simple Button:**

import tkinter as tk

def on\_button\_click():

label.config(text="Button Clicked!")

root = tk.Tk()

label = tk.Label(root, text="Press the Button")

button = tk.Button(root, text="Click Me", command=on\_button\_click)

label.pack()

button.pack()

root.mainloop()

**Entry Widget (Input Field):**

import tkinter as tk

def on\_submit():

result\_label.config(text=f"You entered: {entry.get()}")

root = tk.Tk()

entry = tk.Entry(root)

submit\_button = tk.Button(root, text="Submit", command=on\_submit)

result\_label = tk.Label(root, text="")

entry.pack()

submit\_button.pack()

result\_label.pack()

root.mainloop()

**Checkboxes:**

import tkinter as tk

def on\_checkbox\_change():

result\_label.config(text=f"Selected Options: {', '.join(selected\_options)}")

root = tk.Tk()

selected\_options = []

option1 = tk.StringVar()

option2 = tk.StringVar()

checkbox1 = tk.Checkbutton(root, text="Option 1", variable=option1, command=on\_checkbox\_change)

checkbox2 = tk.Checkbutton(root, text="Option 2", variable=option2, command=on\_checkbox\_change)

result\_label = tk.Label(root, text="")

checkbox1.pack()

checkbox2.pack()

result\_label.pack()

root.mainloop()

**Basic Text Editor:**

import tkinter as tk

from tkinter import filedialog

def open\_file():

file\_path = filedialog.askopenfilename()

with open(file\_path, 'r') as file:

text.delete('1.0', tk.END)

text.insert(tk.END, file.read())

def save\_file():

file\_path = filedialog.asksaveasfilename(defaultextension=".txt")

with open(file\_path, 'w') as file:

file.write(text.get('1.0', tk.END))

root = tk.Tk()

root.title("Basic Text Editor")

menu = tk.Menu(root)

root.config(menu=menu)

file\_menu = tk.Menu(menu)

menu.add\_cascade(label="File", menu=file\_menu)

file\_menu.add\_command(label="Open", command=open\_file)

file\_menu.add\_command(label="Save", command=save\_file)

file\_menu.add\_separator()

file\_menu.add\_command(label="Exit", command=root.quit)

text = tk.Text(root)

text.pack()

root.mainloop()

**Button Click Event:**

import tkinter as tk

# Function to handle button click

def on\_button\_click():

label.config(text="Button Clicked!")

root = tk.Tk()

root.title("Button Example")

label = tk.Label(root, text="Press the button")

label.pack()

button = tk.Button(root, text="Click Me", command=on\_button\_click)

button.pack()

root.mainloop()