# PSP Time Recording Log

# Student: Ranithu Dinsara Ranasinghe

# Module/Function Name: Introduction to Mobile Applications Development - CSP2108.1

File Name: welcome\_screen.lua

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Start Date and Time** | **Stop Date and Time** | **Time Taken** | **LOC** | **Paste actual code here** |
| 08/04/2024 – 1PM | /04/2024 – 2.20PM | 1 Hour  and 20 min | 24 | local composer = require("composer")  local widget = require("widget")  local scene = composer.newScene()  local function onStartButtonPress(event)      composer.gotoScene("graph\_scene", { effect = "slideLeft", time = 500 })  end  function scene:create(event)      local sceneGroup = self.view      local background = display.newRect(display.contentCenterX, display.contentCenterY, display.contentWidth, display.contentHeight)      background:setFillColor(0.7, 0.7, 1)      sceneGroup:insert(background)      local welcomeImage = display.newImage("kNN\_pic2.png")      welcomeImage.width = 200      welcomeImage.height = 200      welcomeImage.x = display.contentCenterX      welcomeImage.y = display.contentCenterY - 25      sceneGroup:insert(welcomeImage)      local titleLabel = display.newText {          text = "kNN Classification App",          x = display.contentCenterX,          y = display.contentCenterY - 170,          fontSize = 18,          font = native.systemFontBold,          shadow = { x = 2, y = 2, blur = 3, color = { 0, 0, 0, 0.6 } }      }      titleLabel:setFillColor(0)      sceneGroup:insert(titleLabel) |
| 10/04/2024 – 7PM | 10/04/2024 – 7.25PM | 25 min | 16 | local startButton = widget.newButton({          width = 200,          height = 60,          shape = "roundedRect",          cornerRadius = 10,          label = "START!!!",          fontSize = 20,          fillColor = { default={0.5, 0.5, 0.5}, over={0.8, 0.8, 0.8} },          labelColor = { default={1, 1, 1} },          onPress = onStartButtonPress      })      startButton.x = display.contentCenterX      startButton.y = display.contentCenterY + 160      sceneGroup:insert(startButton)  end  scene:addEventListener("create", scene)  return scene |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

File Name: graph\_screen.lua

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Start Date and Time** | **Stop Date and Time** | **Time Taken** | **LOC** | **Paste actual code here** |
| 16/04/2024 – 10PM | 16/04/2024 – 10.40PM | 40 min | 33 | local composer = require("composer")  local widget = require("widget")  local scene = composer.newScene()  local function calculateDistance(x1, y1, x2, y2)  return math.sqrt((x2 - x1)^2 + (y2 - y1)^2)  end  local function onPlotButtonPress()  local path = system.pathForFile("knn.csv", system.ResourceDirectory)  local file = io.open(path, "r")  if file then  local distances = {}  for line in file:lines() do  local x, y, label = line:match("([^,]+),([^,]+),([^,]+)")  local distance = calculateDistance(4, 4, tonumber(x), tonumber(y))  table.insert(distances, {label = label, distance = distance})  end  io.close(file)  table.sort(distances, function(a, b) return a.distance < b.distance end)  print("File open:")  print("Euclidean distances from [4,4]:")  for i, data in ipairs(distances) do  print("Point " .. i .. ": Distance = " .. data.distance .. ", Class - " .. data.label)  end |
| 17/04/2024 – 3.25PM | 17/04/2024 – 4.25PM | 1 Hour | 49 | print("\n")  print("\n")  print("\n")  local kValues = {3, 5, 9}  for \_, k in ipairs(kValues) do  print("kNN results for k = " .. k .. " with equal weights:")  local counts = {}  for i = 1, k do  local label = distances[i].label  counts[label] = (counts[label] or 0) + 1  end  local mostFrequentClass  local maxCount = 0  for label, count in pairs(counts) do  if count > maxCount then  mostFrequentClass = label  maxCount = count  end  end  print("Class - " .. mostFrequentClass)  end  else  print("Failed to open the CSV file")  end  end  function scene:create(event)  local sceneGroup = self.view  local background = display.newRect(display.contentCenterX, display.contentCenterY, display.contentWidth, display.contentHeight)  background:setFillColor(0.7, 0.7, 1)  sceneGroup:insert(background)  local plotButton = widget.newButton({  width = 200,  height = 60,  shape = "roundedRect",  cornerRadius = 10,  label = "Week 08",  fontSize = 20,  fillColor = { default={1,1,1}, over={0.8,0.8,0.8} },  onPress = onPlotButtonPress  })  plotButton.x = display.contentCenterX  plotButton.y = display.contentCenterY  sceneGroup:insert(plotButton)  end  scene:addEventListener("create", scene)  return scene |
|  |  |  |  |  |
|  |  |  |  |  |

File Name: choose\_point\_scene.lua

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Start Date and Time** | **Stop Date and Time** | **Time Taken** | **LOC** | **Paste actual code here** |
| 16/04/2024 – 10PM | 16/04/2024 – 10.45PM | 45 min | 33 | local composer = require("composer")  local widget = require("widget")  local scene = composer.newScene()  local function calculateDistance(x1, y1, x2, y2)  return math.sqrt((x2 - x1)^2 + (y2 - y1)^2)  end  local function onPlotButtonPress()  local path = system.pathForFile("knn.csv", system.ResourceDirectory)  local file = io.open(path, "r")  if file then  local distances = {}  for line in file:lines() do  local x, y, label = line:match("([^,]+),([^,]+),([^,]+)")  local distance = calculateDistance(4, 4, tonumber(x), tonumber(y))  table.insert(distances, {label = label, distance = distance})  end  io.close(file)  table.sort(distances, function(a, b) return a.distance < b.distance end)  print("File open:")  print("Euclidean distances from [4,4]:")  for i, data in ipairs(distances) do  print("Point " .. i .. ": Distance = " .. data.distance .. ", Class - " .. data.label)  end |
| 17/04/2024 – 4PM | 17/04/2024 – 4.25PM | 25 min | 49 | print("\n")  print("\n")  print("\n")  local kValues = {3, 5, 9}  for \_, k in ipairs(kValues) do  print("kNN results for k = " .. k .. " with equal weights:")  local counts = {}  for i = 1, k do  local label = distances[i].label  counts[label] = (counts[label] or 0) + 1  end  local mostFrequentClass  local maxCount = 0  for label, count in pairs(counts) do  if count > maxCount then  mostFrequentClass = label  maxCount = count  end  end  print("Class - " .. mostFrequentClass)  end  else  print("Failed to open the CSV file")  end  end  function scene:create(event)  local sceneGroup = self.view  local background = display.newRect(display.contentCenterX, display.contentCenterY, display.contentWidth, display.contentHeight)  background:setFillColor(0.7, 0.7, 1)  sceneGroup:insert(background)  local plotButton = widget.newButton({  width = 200,  height = 60,  shape = "roundedRect",  cornerRadius = 10,  label = "Week 08",  fontSize = 20,  fillColor = { default={1,1,1}, over={0.8,0.8,0.8} },  onPress = onPlotButtonPress  })  plotButton.x = display.contentCenterX  plotButton.y = display.contentCenterY  sceneGroup:insert(plotButton)  end  scene:addEventListener("create", scene)  return scene |
| 24/04/2024 – 11AM | 24/04/2024 – 11.40AM | 40 min | 24 | local composer = require("composer")  local scene = composer.newScene()  local function readCSV(filename)  local coordinates = {}  local path = system.pathForFile(filename, system.ResourceDirectory)  local file, errorString = io.open(path, "r")  if not file then  print("File error: " .. errorString)  else  print("File open: ")  for line in file:lines() do  local coordinate\_x, coordinate\_y, coordinate\_name = line:match("(%d+),(%d+),(%a+)")  if coordinate\_x and coordinate\_y and coordinate\_name then  coordinate\_x = math.min(math.max(0, tonumber(coordinate\_x)), 10)  coordinate\_y = math.min(math.max(0, tonumber(coordinate\_y)), 10)  coordinates[#coordinates+1] = { coordinate\_name = coordinate\_name, coordinate\_x = tonumber(coordinate\_x), coordinate\_y = tonumber(coordinate\_y) }  else  print("Error parsing line:", line)  end  end  io.close(file)  end  return coordinates  end |
| 25/04/2024 – 6PM | 25/04/2024 – 7.25PM | 1 Hour and 25 min | 102 | local function calculateDistance(x1, y1, x2, y2)  return math.sqrt((x2 - x1)^2 + (y2 - y1)^2)  end  local function plotGraph(coordinates, testPoint)  local graphGroup = display.newGroup()  local screenWidth = display.contentWidth  local screenHeight = display.contentHeight  local graphWidth = screenWidth - 60  local graphHeight = screenHeight - 60  local marginLeft = 40  local marginTop = 30  local scaleX = graphWidth / 10  local scaleY = graphHeight / 10  local xAxis = display.newLine(graphGroup, marginLeft, marginTop + graphHeight, marginLeft + graphWidth, marginTop + graphHeight)  local yAxis = display.newLine(graphGroup, marginLeft, marginTop, marginLeft, marginTop + graphHeight)  for i = 1, 9 do  local x = marginLeft + i \* scaleX  local y = marginTop + i \* scaleY  local xGridLine = display.newLine(graphGroup, x, marginTop, x, marginTop + graphHeight)  xGridLine:setStrokeColor(0.5)  xGridLine.strokeWidth = 2  local yGridLine = display.newLine(graphGroup, marginLeft, y, marginLeft + graphWidth, y)  yGridLine:setStrokeColor(0.5)  yGridLine.strokeWidth = 2  end  for i = 0, 10, 5 do  local tickX = marginLeft + i \* scaleX  local tickY = marginTop + (10 - i) \* scaleY  local tick = display.newLine(graphGroup, tickX, marginTop + graphHeight - 5, tickX, marginTop + graphHeight + 5)  tick:setStrokeColor(0)  tick.strokeWidth = 2  local labelX = display.newText(graphGroup, tostring(i), tickX, marginTop + graphHeight + 15, native.systemFont, 12)  labelX.anchorY = 0  labelX:setFillColor(0)  local labelY = display.newText(graphGroup, tostring(i), marginLeft - 20, tickY, native.systemFont, 12)  labelY.anchorX = 1  labelY:setFillColor(0)  end  local distances = {}  for \_, coordinate in pairs(coordinates) do  local distance = calculateDistance(coordinate.coordinate\_x, coordinate.coordinate\_y, testPoint[1], testPoint[2])  table.insert(distances, {distance = distance, class = coordinate.coordinate\_name})  end  table.sort(distances, function(a, b) return a.distance < b.distance end)  local nearestNeighbors = {}  for i = 1, 3 do  table.insert(nearestNeighbors, distances[i].class)  end  local classCounts = {a = 0, b = 0, c = 0}  for \_, class in ipairs(nearestNeighbors) do  classCounts[class] = classCounts[class] + 1  end  local majorityClass  local maxCount = 0  for class, count in pairs(classCounts) do  if count > maxCount then  majorityClass = class  maxCount = count  end  end  local color  if majorityClass == "a" then  color = "(Blue)"  elseif majorityClass == "b" then  color = "(Red)"  else  color = "(Yellow)"  end  print("Test point [6,5] belongs to class:", majorityClass, color)  for \_, coordinate in pairs(coordinates) do  local classColor  if coordinate.coordinate\_name == "a" then  classColor = {0, 0, 1}  elseif coordinate.coordinate\_name == "b" then  classColor = {1, 0, 0}  else  classColor = {1, 1, 2}  end  local circle = display.newCircle(graphGroup, marginLeft + coordinate.coordinate\_x \* scaleX, marginTop + (10 - coordinate.coordinate\_y) \* scaleY, 5)  circle:setFillColor(unpack(classColor))  end  scene.view:insert(graphGroup)  end  function scene:create(event)  local sceneGroup = self.view  local background = display.newRect(display.contentCenterX, display.contentCenterY, display.contentWidth, display.contentHeight)  background:setFillColor(0.7, 0.7, 1)  sceneGroup:insert(background)  local coordinates = readCSV("knn.csv")  if #coordinates == 0 then  print("The table is empty, there is no valid data in the file.")  else  for \_, coordinate in pairs(coordinates) do  print(coordinate.coordinate\_name, coordinate.coordinate\_x, coordinate.coordinate\_y)  end  end  local testPoint = {6, 5} -- Test point [6,5]  plotGraph(coordinates, testPoint)  end  scene:addEventListener("create", scene)  return scene |
|  |  |  |  |  |
|  |  |  |  |  |

File Name: choose\_point\_scene.lua

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Start Date and Time** | **Stop Date and Time** | **Time Taken** | **LOC** | **Paste actual code here** |
| 30/04/2024 – 8PM | 30/04/2024 – 9.20PM | 1 Hour  and 20 min | 99 | local composer = require("composer")  local scene = composer.newScene()  local x1Field, y1Field, kField  local function classifyPoint(x, y, k)      local sum = x + y      if sum < 10 then          return "Class A"      elseif sum < 20 then          return "Class B"      else          return "Class C"      end  end  local function onChooseButtonTap(event)      local x1 = tonumber(x1Field.text) or 0      local y1 = tonumber(y1Field.text) or 0      local k = tonumber(kField.text) or 1      local metric = selectedMetric      local useHarmonic = useHarmonicWeighting      composer.gotoScene("choose\_metrics\_scene", {          effect = "slideLeft",          time = 500,          params = {              testPoints = {{x1, y1}},              k = k,              metric = metric,              useHarmonic = useHarmonic          }      })  end  local function onBackButtonTap(event)      composer.gotoScene("graph\_scene", {effect = "slideRight", time = 500})  end  function scene:create(event)      local sceneGroup = self.view      local background = display.newRect(display.contentCenterX, display.contentCenterY, display.contentWidth, display.contentHeight)      background:setFillColor(0.7, 0.7, 1)      sceneGroup:insert(background)      local title = display.newText {          text = "Choose Point to Test",          x = display.contentCenterX,          y = 50,          fontSize = 18,          font = native.systemFontBold      }      title:setFillColor(1)      sceneGroup:insert(title)      x1Field = native.newTextField(display.contentCenterX, 140, 60, 30)      x1Field.placeholder = "x1"      sceneGroup:insert(x1Field)      y1Field = native.newTextField(display.contentCenterX, 190, 60, 30)      y1Field.placeholder = "y1"      sceneGroup:insert(y1Field)      kField = native.newTextField(display.contentCenterX, 240, 80, 30)      kField.placeholder = "k"      sceneGroup:insert(kField)      local buttonWidth = 150      local buttonHeight = 30      local button = display.newRoundedRect(          sceneGroup,          display.contentCenterX,          display.contentHeight - 130,          buttonWidth,          buttonHeight,          8      )      button:setFillColor(0.5, 0.5, 0.5)      local chooseButton = display.newText {          text = "Choose",          x = display.contentCenterX,          y = display.contentHeight - 130,          fontSize = 18,          font = native.systemFont      }      chooseButton:setFillColor(1)      sceneGroup:insert(chooseButton)      local backButton = display.newRect(display.contentWidth - 160, display.contentHeight - 35, 60, 30)      backButton:setFillColor(0.5, 0.5, 0.5)      sceneGroup:insert(backButton)      local backButtonText = display.newText {          text = "Back",          x = backButton.x,          y = backButton.y,          fontSize = 14,          font = native.systemFont      }      sceneGroup:insert(backButtonText)      backButton:addEventListener("tap", onBackButtonTap)      chooseButton:addEventListener("tap", onChooseButtonTap)  end  scene:addEventListener("create", scene)  return scene |

File Name: choose\_metrics\_scene.lua

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Start Date and Time** | **Stop Date and Time** | **Time Taken** | **LOC** | **Paste actual code here** |
| 01/05/2024 – 10AM | 01/05/2024 – 12.15 | 2 hours and 15 min | 169 | local composer = require("composer")  local widget = require("widget")  local scene = composer.newScene()  local selectedMetric = "Euclidean"  local useHarmonicWeighting = false  local function onMetricRadioButtonTap(event)      selectedMetric = event.target.id      print("Selected Metric:", selectedMetric)  end  local function onHarmonicWeightingCheckboxTap(event)      useHarmonicWeighting = event.target.isOn      print("Use Harmonic Weighting:", useHarmonicWeighting)  end  local function calculateClassificationResult()          local classes = {"Class A", "Class B"}        local randomIndex = math.random(1, #classes)      return classes[randomIndex]  end  -- Function to handle "Back" button tap event  local function onBackButtonTap(event)      composer.gotoScene("choose\_point\_scene", { effect = "slideRight", time = 500 })  end  -- Function to handle "Next" button tap event  local function onNextButtonTap(event)      -- Calculate the classification result      local classificationResult = calculateClassificationResult()      -- Navigate to the next scene and pass the classification result as a parameter      composer.gotoScene("classification\_results\_scene", {          effect = "slideLeft",          time = 500,          params = { classificationResult = classificationResult }      })  end  function scene:create(event)      local sceneGroup = self.view      -- Background      local background = display.newRect(display.contentCenterX, display.contentCenterY, display.contentWidth, display.contentHeight)      background:setFillColor(0.7, 0.7, 1)      sceneGroup:insert(background)      -- Title      local title = display.newText {          text = "Choose Metrics",          x = display.contentCenterX,          y = 50,          fontSize = 18,          font = native.systemFontBold      }      title:setFillColor(1)      sceneGroup:insert(title)      -- Metric radio buttons      local metricOption1 = widget.newSwitch {          id = "Euclidean",          style = "radio",          left = display.contentCenterX - 90,          top = 106,          initialSwitchState = true,          onPress = onMetricRadioButtonTap      }      sceneGroup:insert(metricOption1)      local option1Label = display.newText {          text = "Euclidean",          x = display.contentCenterX - 18,          y = 120,          fontSize = 16,          align = "left"      }      sceneGroup:insert(option1Label)      local metricOption2 = widget.newSwitch {          id = "Manhattan",          style = "radio",          left = display.contentCenterX - 90,          top = 145,          initialSwitchState = false,          onPress = onMetricRadioButtonTap      }      sceneGroup:insert(metricOption2)      local option2Label = display.newText {          text = "Manhattan",          x = display.contentCenterX - 17,          y = 160,          fontSize = 16,          align = "left"      }      sceneGroup:insert(option2Label)      -- Harmonic weighting checkbox      local harmonicWeightingCheckbox = widget.newSwitch {          x = display.contentCenterX - 78,          y = 200,          style = "checkbox",          id = "HarmonicWeightingCheckbox",          onPress = onHarmonicWeightingCheckboxTap      }      sceneGroup:insert(harmonicWeightingCheckbox)      local checkboxLabel = display.newText {          text = "       Harmonic Weighting",          x = display.contentCenterX - 0.1,          y = 200,          fontSize = 16,          align = "right"      }      sceneGroup:insert(checkboxLabel)      local buttonWidth = 150      local buttonHeight = 30      local button = display.newRoundedRect(          sceneGroup,          display.contentCenterX,          display.contentHeight - 230,          buttonWidth,          buttonHeight,          8      )      -- Next button      local nextButton = widget.newButton {          label = "Next",          x = display.contentCenterX,          y = 250,          width = 150,          height = 40,          fontSize = 16,          onRelease = onNextButtonTap      }      sceneGroup:insert(nextButton)      -- Back button      local backButton = display.newRect(display.contentWidth - 160, display.contentHeight - 35, 60, 30)      backButton:setFillColor(0.5, 0.5, 0.5)      sceneGroup:insert(backButton)      local backButtonText = display.newText {          text = "Back",          x = backButton.x,          y = backButton.y,          fontSize = 14,          font = native.systemFont      }      sceneGroup:insert(backButtonText)      backButton:addEventListener("tap", onBackButtonTap)  end  scene:addEventListener("create", scene)  return scene |

File Name: classification\_results\_scene.lua

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Start Date and Time** | **Stop Date and Time** | **Time Taken** | **LOC** | **Paste actual code here** |
| 02/05/2024 – 9.00PM | 2/05/2024 – 9.30PM | 30 min | 49 | local composer = require("composer")  local scene = composer.newScene()  local function onBackButtonTap(event)      composer.gotoScene("choose\_metrics\_scene", { effect = "slideRight", time = 500 })  end  function scene:create(event)      local sceneGroup = self.view        local background = display.newRect(display.contentCenterX, display.contentCenterY, display.contentWidth, display.contentHeight)      background:setFillColor(0.7, 0.7, 1)      sceneGroup:insert(background)      local classificationResult = event.params.classificationResult        local resultLabel = display.newText {          text = "Classification Result: " .. classificationResult,          x = display.contentCenterX,          y = display.contentCenterY,          fontSize = 15,          font = native.systemFontBold      }      sceneGroup:insert(resultLabel)      local backButton = display.newRect(display.contentWidth - 160, display.contentHeight - 35, 60, 30)      backButton:setFillColor(0.5, 0.5, 0.5)      sceneGroup:insert(backButton)      local backButtonText = display.newText {          text = "Back",          x = backButton.x,          y = backButton.y,          fontSize = 14,          font = native.systemFont      }      sceneGroup:insert(backButtonText)      backButton:addEventListener("tap", onBackButtonTap)  end  scene:addEventListener("create", scene)  return scene |

File Name: unit-tests.lua and myFunctions.lua

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Start Date and Time** | **Stop Date and Time** | **Time Taken** | **LOC** | **Paste actual code here** |
| 03/05/2024 – 8.00PM | 03/05/2024 – 9.45PM | 1 Hour  and 45 min | 30 | -- Import the necessary modules for unit testing  local lunatest = require("lunatest")  module(..., package.seeall)  local myFunctions = require("modules.myFunctions")  function testCalculateEuclideanDistance()      assert(math.abs(myFunctions.calculateEuclideanDistance({0, 0}, {3, 4}) - 5) < 0.0001, "Euclidean distance calculation failed for (0,0) to (3,4)")      assert(math.abs(myFunctions.calculateEuclideanDistance({-3, -4}, {0, 0}) - 5) < 0.0001, "Euclidean distance calculation failed for (-3,-4) to (0,0)")      assert(myFunctions.calculateEuclideanDistance({3, 0}, {0, 4}) == 5, "Euclidean distance calculation failed for (3,0) to (0,4)")      assert(myFunctions.calculateEuclideanDistance({0, 0}, {0, 0}) == 0, "Euclidean distance calculation failed for identical points")  end  function testCalculateManhattanDistance()      assert(myFunctions.calculateManhattanDistance({0, 0}, {3, 4}) == 7, "Manhattan distance calculation failed for (0,0) to (3,4)")      assert(myFunctions.calculateManhattanDistance({-3, -4}, {0, 0}) == 7, "Manhattan distance calculation failed for (-3,-4) to (0,0)")      assert(myFunctions.calculateManhattanDistance({3, 0}, {0, 4}) == 7, "Manhattan distance calculation failed for (3,0) to (0,4)")      assert(myFunctions.calculateManhattanDistance({0, 0}, {0, 0}) == 0, "Manhattan distance calculation failed for identical points")  end  function testOnHarmonicWeightingCheckboxTap()        local event = { target = { isOn = true } }      myFunctions.onHarmonicWeightingCheckboxTap(event)      assert(myFunctions.useHarmonicWeighting == true, "Harmonic weighting checkbox tap event failed for true state")      event.target.isOn = false      myFunctions.onHarmonicWeightingCheckboxTap(event)      assert(myFunctions.useHarmonicWeighting == false, "Harmonic weighting checkbox tap event failed for false state")  end  function testSomethingToSkip()      return "skip"  end  function testAnotherThingToSkip()      return "skip"  end  lunatest.run() |
|  |  |  |  |  |
| 03/05/2024 – 9.00PM | 03/05/2024 – 9.45PM | 45 min | 17 | local lunatest = require("lunatest")  -- Define the test suite  module(..., package.seeall)  local myFunctions = {}  function myFunctions.calculateEuclideanDistance(point1, point2)      local dx = point2[1] - point1[1]      local dy = point2[2] - point1[2]      return math.sqrt(dx \* dx + dy \* dy)  end  function myFunctions.calculateManhattanDistance(point1, point2)      return math.abs(point2[1] - point1[1]) + math.abs(point2[2] - point1[2])  end  function myFunctions.onHarmonicWeightingCheckboxTap(event)      myFunctions.useHarmonicWeighting = event.target.isOn  end  return myFunctions |