In [ ]:
```
#Q-1) Use the file data.csv which contains 169 rows and 4 columns.
# 1. Convert this file into pandas Data Frame and Display basic statistics like mean, std, quartiles, etc. for this dat
# 2. Print first and last 5 rows. Also print the shape of the dataframe.
# 3. Create a correlation table for the data frame and comment about what kind of correlation is there between Duration
# 4. Find whether there any null or NA values, drop all such rows if found in the data frame and print the shape of the
# 5. Prepare a scatter matrix for the following data frame.
# 6. Prepare a parallel coordinates for Duration v/s Pulse, Maxpulse and Calories (all 3 other columns).
# 7. Prepare a cross-tabulation for Duration v/s Pulse.
# 8. Do Maxpulse have any outliers? Find using function
```

In [17]:
```
import pandas as pd
df=pd.read_csv("data.csv")
# print(df)
print(df.describe())
```

```
         Duration       Pulse    Maxpulse     Calories
count  169.000000  169.000000  169.000000   164.000000
mean    63.846154  107.461538  134.047337   375.790244
std     42.299949   14.510259   16.450434   266.379919
min     15.000000   80.000000  100.000000    50.300000
25%     45.000000  100.000000  124.000000   250.925000
50%     60.000000  105.000000  131.000000   318.600000
75%     60.000000  111.000000  141.000000   387.600000
max    300.000000  159.000000  184.000000  1860.400000
```

In [7]:
```
print(df.head(5))
print(df.shape)
```

```
   Duration  Pulse  Maxpulse  Calories
0        60    110       130     409.1
1        60    117       145     479.0
2        60    103       135     340.0
3        45    109       175     282.4
4        45    117       148     406.0
(169, 4)
```

In [10]:
```
print(df.corr())
print("duration and calories have strong relationship")
```

```
          Duration     Pulse  Maxpulse  Calories
Duration  1.000000 -0.155408  0.009403  0.922717
Pulse    -0.155408  1.000000  0.786535  0.025121
Maxpulse  0.009403  0.786535  1.000000  0.203813
Calories  0.922717  0.025121  0.203813  1.000000
duration and calories have strong relationship
```

In [18]:
```
# Find whether there any null or NA values, drop all such rows if found in the data frame and print the shape of the da
df.isna().sum()
```

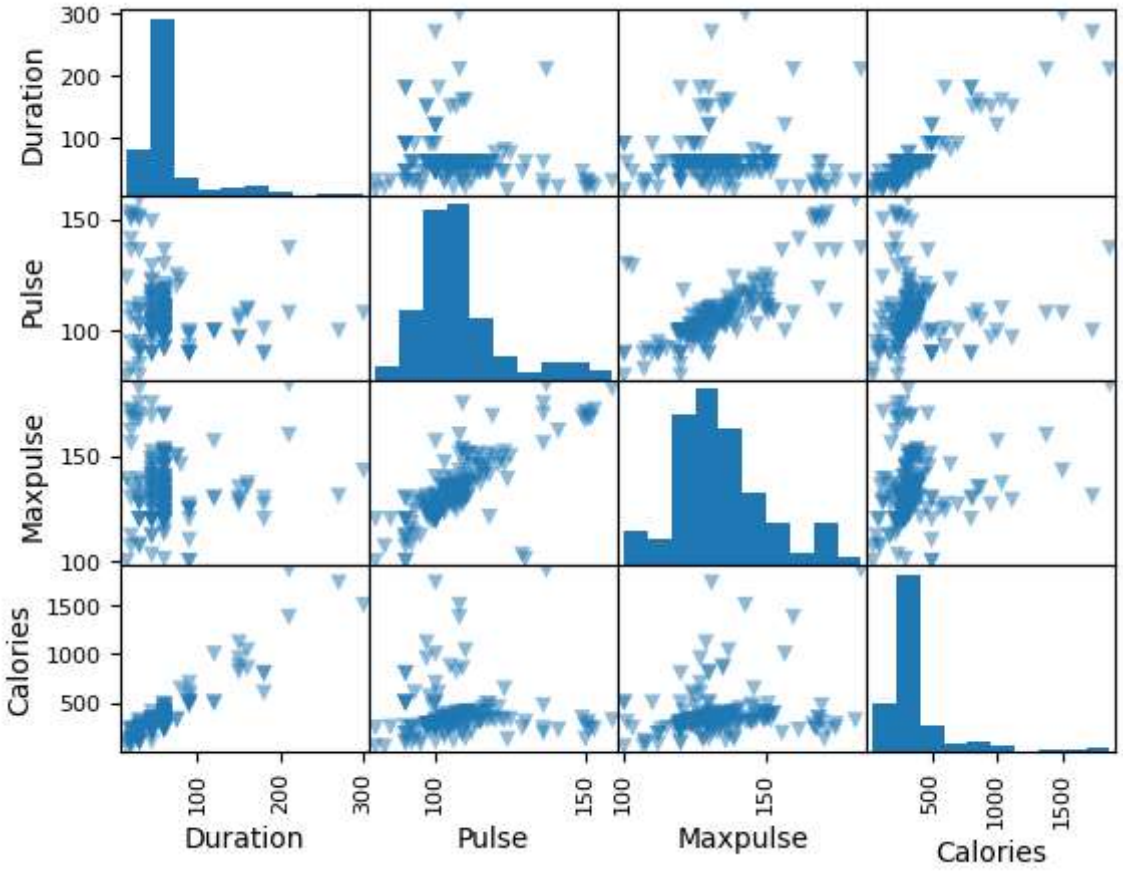
Out[18]:
```
Duration    0
Pulse       0
Maxpulse    0
Calories    5
dtype: int64
```

In [19]:
```
df=df.dropna()
```

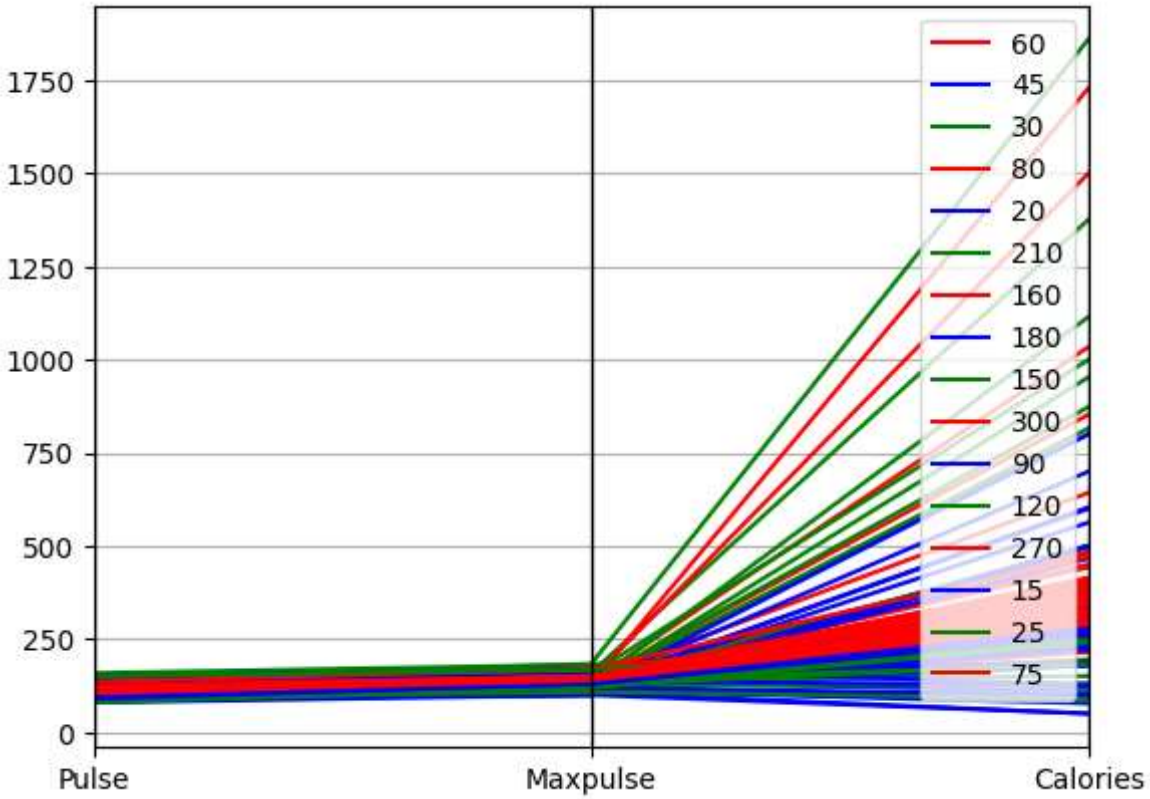
In [20]:
```
df.isna().sum()
```

Out[20]:
```
Duration    0
Pulse       0
Maxpulse    0
Calories    0
dtype: int64
```

In [24]:
```
# 5. Prepare a scatter matrix for the following data frame.
import matplotlib.pyplot as plt
pd.plotting.scatter_matrix(df,marker='v',alpha=0.5)
plt.show()
```



In [25]:
```
# 6. Prepare a parallel coordinates for Duration v/s Pulse, Maxpulse and Calories (all 3 other columns).
from pandas.plotting import parallel_coordinates
pl=parallel_coordinates(df,'Duration',cols=['Pulse','Maxpulse','Calories'],color=['red','blue','green'])
```



In [27]:
```
# 7. Prepare a cross-tabulation for Duration v/s Pulse.
pd.crosstab(df['Duration'],df['Pulse'],rownames=['Duration'],colnames=['Pulse'])
```

Out[27]:

Pulse	80	83	85	90	92	93	95	97	98	99	...	130	136	137	141	149	150	151	152	153	159
Duration																					
15	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
20	0	1	0	0	0	0	1	0	0	0	...	0	1	0	1	0	1	1	0	1	0
25	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	1	0	0
30	1	0	1	2	1	1	1	0	0	0	...	0	1	0	0	0	1	1	0	0	1
45	0	0	0	3	0	0	1	1	0	0	...	0	0	0	0	1	0	0	0	0	0
60	0	0	0	0	2	1	0	3	5	1	...	1	1	0	0	0	0	0	0	0	0
75	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
80	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
90	0	0	0	4	0	1	0	0	1	1	...	0	0	0	0	0	0	0	0	0	0
120	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
150	0	0	0	0	0	0	0	2	0	0	...	0	0	0	0	0	0	0	0	0	0
160	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
180	0	0	0	2	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
210	0	0	0	0	0	0	0	0	0	0	...	0	0	1	0	0	0	0	0	0	0
270	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
300	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

16 rows × 47 columns

In [28]:
```
# 8.Do Maxpulse have any outliers? Find using function
def find_outliers(ds, col):
    quart1 = ds[col].quantile(0.25)
    quart3 = ds[col].quantile(0.75)
    IQR = quart3 - quart1 #Inter-quartile range
    low_val = quart1 - 1.5*IQR
    high_val = quart3 + 1.5*IQR
    print(low_val)
    print(high_val)
    ds = ds.loc[(ds[col] < low_val) | (ds[col] > high_val)]
    return ds
find_outliers(df,'Maxpulse')
```

```
95.5
171.5
```

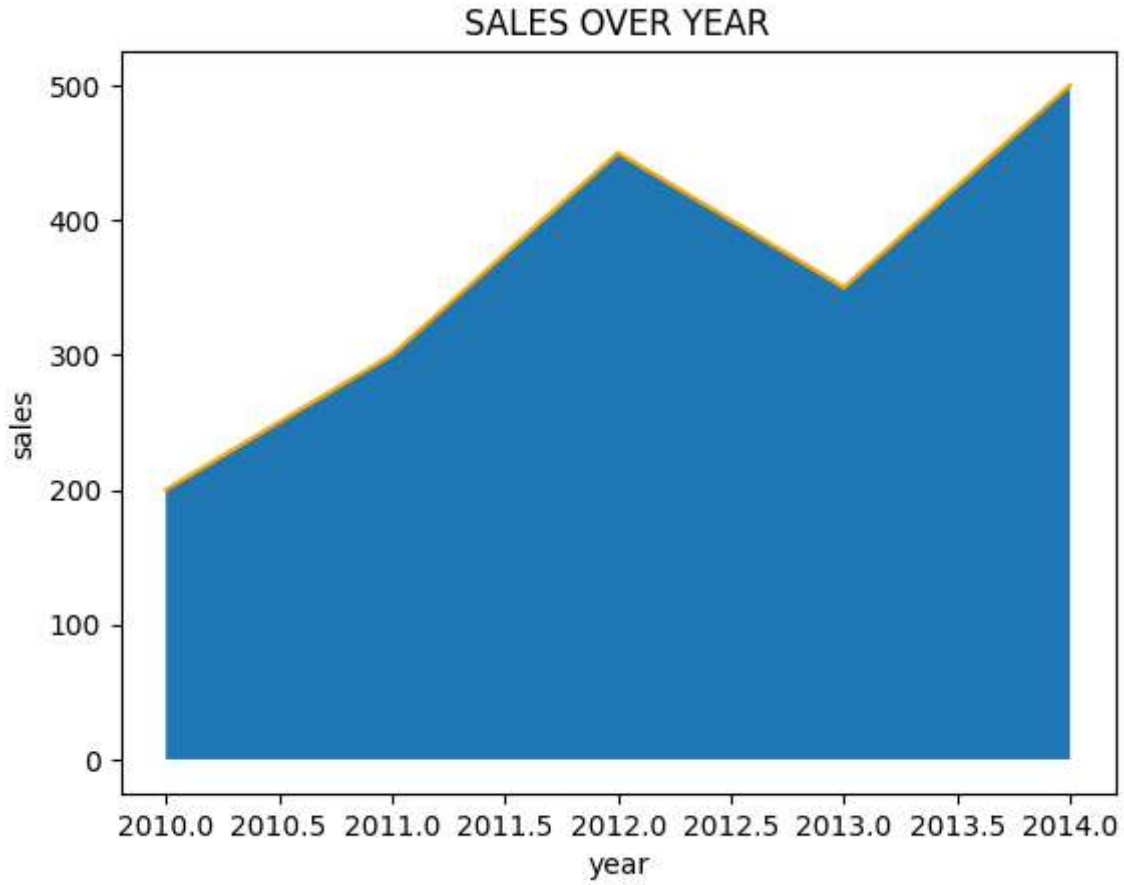
Out[28]:

	Duration	Pulse	Maxpulse	Calories
3	45	109	175	282.4
54	30	136	175	238.0
58	20	153	172	226.4
80	30	159	182	319.2
109	210	137	184	1860.4

In [9]:
```
import random
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

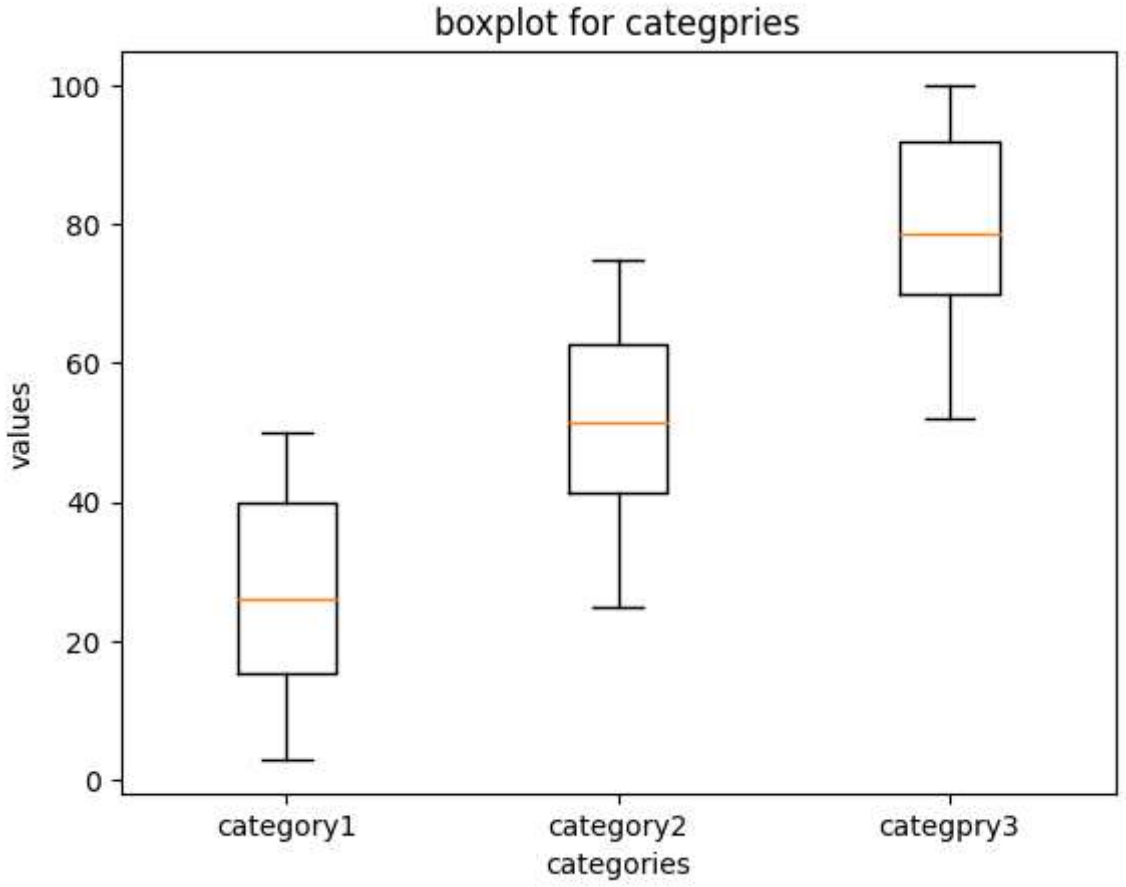
In [1]:
```
# Q-2) random
# # Sample data for area plot
# years = [2010, 2011, 2012, 2013, 2014]
# sales = [200, 300, 450, 350, 500]
# # Sample data for box plot
# category1 = [random.randint(1, 50) for _ in range(50)]
# category2 = [random.randint(25, 75) for _ in range(50)]
# category3 = [random.randint(50, 100) for _ in range(50)]
# # Sample data for scatter plot
# x = [random.uniform(0, 10) for _ in range(50)]
# y = [random.uniform(0, 10) for _ in range(50)]
# # Sample data for heatmap
# import numpy as np
# data = np.random.rand(5, 5)
# # Sample data for regression plot
# height = [160, 165, 170, 175, 180, 185]
# weight = [60, 65, 70, 75, 80, 85]
# Use the above code to generate sample data and then create the following:
# 1. Using the sample data for years and sales, create an area plot to visualize the trend in sales over the years. Wha
# 2. Utilizing the data in category1, category2, and category3, create a box plot using Matplotlib. How does the box pl
# (answer as a comment)?
# 3. Using the generated data for x and y, create a scatter plot with Matplotlib. What patterns or correlations, if any
# comment)?
# 4. Employ the sample data to create a heatmap using Seaborn. What does the heatmap convey about the relationships bet
# 5. With the height and weight data, generate a regression plot using Seaborn. What conclusions can be drawn about the
# comment)?
```

In [8]:
```
#task-1
import matplotlib.pyplot as plt
years = [2010, 2011, 2012, 2013, 2014]
sales = [200, 300, 450, 350, 500]
plt.fill_between(years,sales)
plt.plot(years,sales,color='orange')
plt.title('SALES OVER YEAR')
plt.xlabel('year')
plt.ylabel('sales')
plt.show()
```



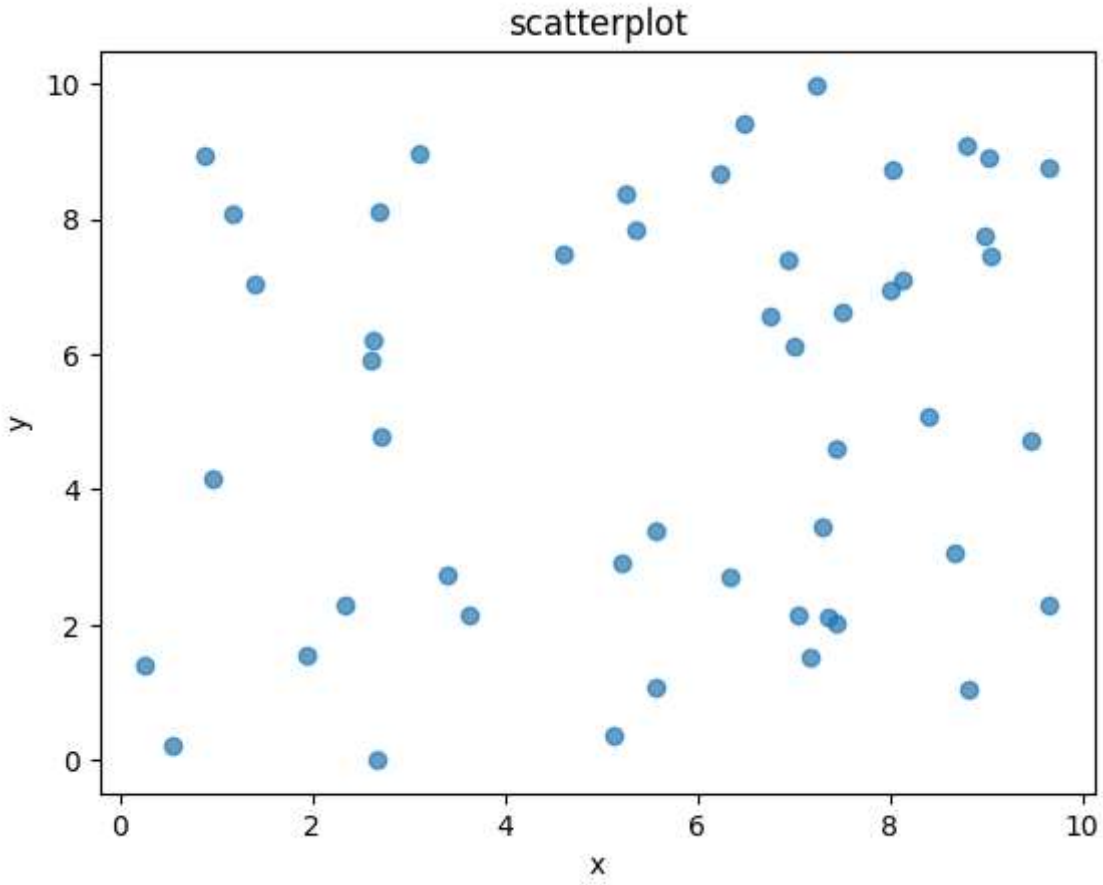
In [12]:
```
#task-2
# import random

category1 = [random.randint(1, 50) for _ in range(50)]
category2 = [random.randint(25, 75) for _ in range(50)]
category3 = [random.randint(50, 100) for _ in range(50)]
data=[category1,category2,category3]
plt.boxplot(data,labels=['category1','category2','categpry3'])
plt.title('boxplot for categpries')
plt.xlabel('categories')
plt.ylabel('values')
plt.show()
```

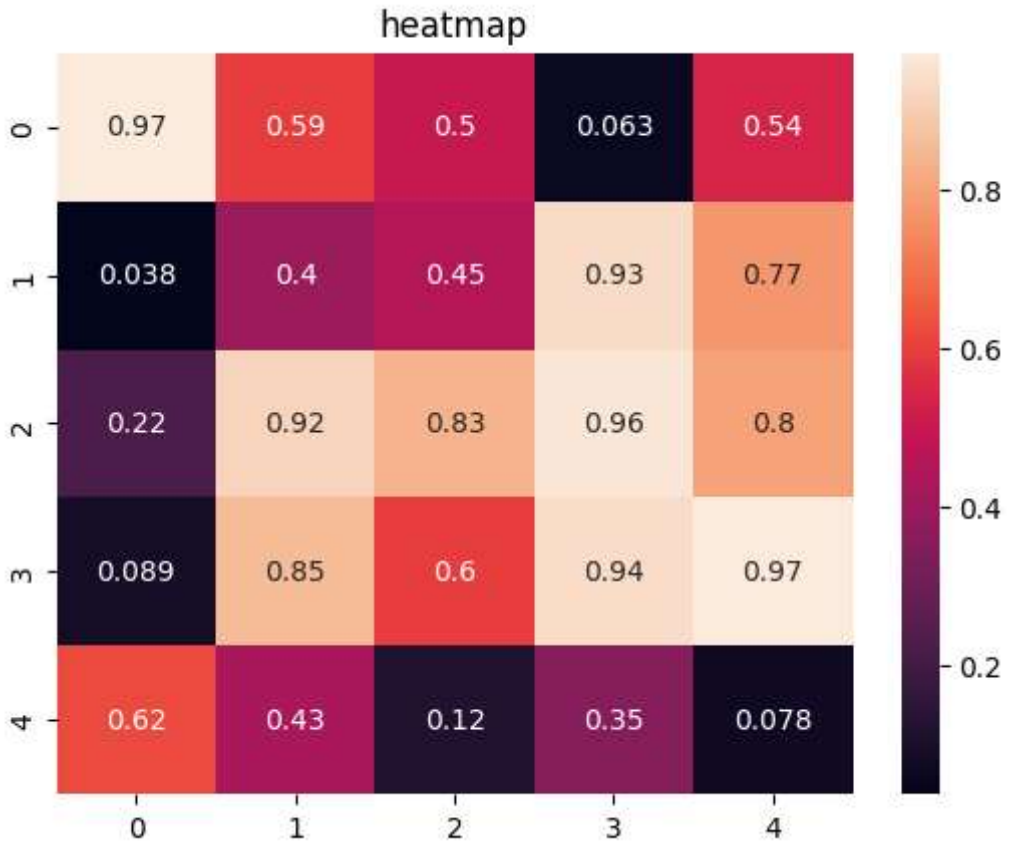


In [14]:
```
#task-3
x = [random.uniform(0, 10) for _ in range(50)]
y = [random.uniform(0, 10) for _ in range(50)]
plt.scatter(x,y,alpha=0.7)
plt.title('scatterplot')
plt.xlabel('x')
plt.ylabel('y')
plt.plot()
```

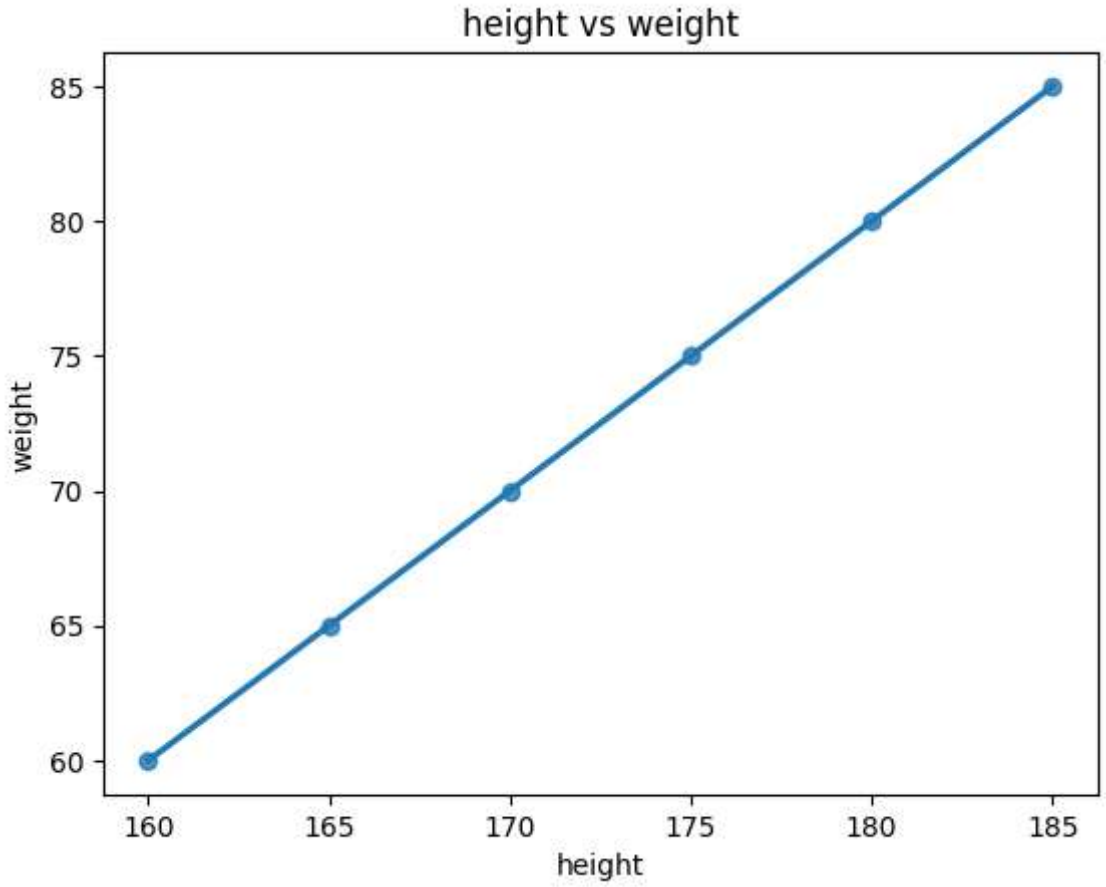
Out[14]: []



In [17]:
```
#task-4
data = np.random.rand(5, 5)
sns.heatmap(data,annot=True)
plt.title('heatmap')
plt.show()
```



In [20]:
```
#task-5
height = [160, 165, 170, 175, 180, 185]
weight = [60, 65, 70, 75, 80, 85]
sns.regplot(x=height,y=weight,scatter=True)
plt.title('height vs weight')
plt.xlabel('height')
plt.ylabel('weight')
plt.show()
```



In [21]:

```python
# Q-3
import re

def is_valid_email(email):
    pattern = r'^[\w.%+-]+@[\w.-]+\.[a-zA-Z]{2,}$'
    if re.match(pattern, email):
        return True
    else:
        return False
def is_strong_password(password):
    pattern = r'^(?=.*[A-Z])(?=.*[a-z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{8,}$'
    if re.match(pattern, password):
        return True
    else:
        return False
import re

def extract_phone_numbers(text):
    pattern = r'\+?(\d{2})?[-.\s]?(\d{10})'
    phone_numbers = re.findall(pattern, text)
    formatted_numbers = ['+91-' + number[1] if number[0] == '' else '+91-' + number[0] + '-' + number[1] for number in

    return formatted_numbers
text = "Here are some phone numbers: +91 1234567890, 9876543210, 080-12345678, +91-9876543210"
phone_numbers = extract_phone_numbers(text)
print("Extracted Phone Numbers:")
for number in phone_numbers:
    print(number)

password = input("Enter a password: ")
if is_strong_password(password):
    print("Strong password")
else:
    print("Weak password")
email = input("Enter an email address: ")
if is_valid_email(email):
    print("Valid email address")
else:
    print("Invalid email address")
```

```
Extracted Phone Numbers:
+91-91-1234567890
+91-9876543210
+91-91-9876543210
Weak password
Invalid email address
```

In [1]:

```python
#Q-4
import re

def extract_urls(text):
    url_pattern = r'http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\\(\\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+'
    urls = re.findall(url_pattern, text)
    return urls
def is_valid_ipv4(ip):
    ipv4_pattern = r'^\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}$'
    return re.match(ipv4_pattern, ip) is not None
def extract_html_tags(html_text):
    html_tags = re.findall(r'<[^>]+>', html_text)
    return html_tags

html_document = "<html><head><title>Sample Page</title></head><body><p>This is a <b>sample</b> HTML document.</p></body
tags = extract_html_tags(html_document)
for tag in tags:
    print(tag)
input_ip = "192.0.2.146"
if is_valid_ipv4(input_ip):
    print(f"{input_ip} is a valid IPv4 address.")
else:
    print(f"{input_ip} is not a valid IPv4 address.")
input_text = "Visit my website at https://www.example.com and check out https://www.example2.com."
urls = extract_urls(input_text)
for url in urls:
    print(url)
```

```
<html>
<head>
<title>
</title>
</head>
<body>
<p>
<b>
</b>
</p>
</body>
</html>
192.0.2.146 is a valid IPv4 address.
https://www.example.com (https://www.example.com)
https://www.example2.com. (https://www.example2.com.)
```

In [17]:

```
#Q-5
from bs4 import BeautifulSoup
import pandas as pd
import numpy as np

# Load the HTML file
with open('fakepython.html', 'r', encoding='utf-8') as html_file:
    html_content = html_file.read()

# Create a BeautifulSoup object
soup = BeautifulSoup(html_content, 'html.parser')

# Task 2: Extract and print Python-related job titles
python_job_titles = []
job_elements = soup.find_all('div', class_='column is-half')
# print(job_elements)
for job in job_elements:
    title = job.find('h2').text.strip()
    # print(title)
    if 'python' in title.lower():
        python_job_titles.append(title)

# print(python_job_titles)
job_details = []
for job in job_elements:
    title = job.find('h2').text.strip()
    location = job.find('p', class_='location').text.strip()
    company = job.find('h3').text.strip()
    job_details.append({
        'Title': title,
        'Location': location,
        'Company': company,
    })

# print(job_details)
df = pd.DataFrame(job_details)
python_jobs_df = df[df['Title'].str.lower().str.contains('python')]

# Print the DataFrame
print("\nPython-related jobs DataFrame:")
```

```
print(python_jobs_df)
```

```
Python-related jobs DataFrame:
                                    Title               Location  \
0                   Senior Python Developer        Stewartbury, AA
10                Software Engineer (Python)           Ericberg, AE
20          Python Programmer (Entry-Level)          Port Sara, AE
30          Python Programmer (Entry-Level)        West Victor, AE
40              Software Developer (Python)         Brockburgh, AE
50                        Python Developer   East Michaelfort, AA
60  Back-End Web Developer (Python, Django)     South Kimberly, AA
70  Back-End Web Developer (Python, Django)  New Elizabethside, AA
80          Python Programmer (Entry-Level)      Robertborough, AP
90              Software Developer (Python)      Martinezburgh, AE

                    Company
0   Payne, Roberts and Davis
10                Garcia PLC
20    Moss, Duncan and Allen
30            Cooper and Sons
40               Adams-Brewer
50            Rivera and Sons
60          Stewart-Alexander
70    Jackson, Ali and Mckee
80                Mathews Inc
90          Moreno-Rodriguez
```

In [38]:

```
from bs4 import BeautifulSoup
import pandas as pd

# Load the HTML file
with open('Quotes to Scrape.html', 'r', encoding='utf-8') as html_file:
    html_content = html_file.read()

# Create a BeautifulSoup object
soup = BeautifulSoup(html_content, 'html.parser')

# Task 2: Extract and print all quotes
quote_elements = soup.find_all('span', class_='text')
# print(quote_elements)
quote=[]
author=[]
# Task 3: Extract and print all quotes and authors
quotes = [quote.text for quote in quote_elements]
quote_author_elements = soup.find_all('div',class_='quote')
quote_author_pairs = [(element.find('span', class_='text').text.strip(), element.find('small', class_='author').text.st
# print(quote_author_pairs)
data = {'Quote': quotes, 'Author': [author for _, author in quote_author_pairs]}
df=pd.DataFrame(data)
df
```

Out[38]:

	Quote	Author
0	“The world as we have created it is a process ...	Albert Einstein
1	“It is our choices, Harry, that show what we t...	J.K. Rowling
2	“There are only two ways to live your life. On...	Albert Einstein
3	“The person, be it gentleman or lady, who has ...	Jane Austen
4	“Imperfection is beauty, madness is genius and...	Marilyn Monroe
5	“Try not to become a man of success. Rather be...	Albert Einstein
6	“It is better to be hated for what you are tha...	André Gide
7	“I have not failed. I've just found 10,000 way...	Thomas A. Edison
8	“A woman is like a tea bag; you never know how...	Eleanor Roosevelt
9	“A day without sunshine is like, you know, nig...	Steve Martin

In [52]:
```
# Write a program to create a Model using linear regression to predict the charges of insurance using the csv file prov
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
df=pd.read_csv('insurance.csv')
# print(df.isna().sum())
print("there is no null value")
df=pd.get_dummies(data=df,drop_first=True)
# print(df)
x=df[["age","sex_male","bmi","children","smoker_yes","region_northwest","region_southeast","region_southwest"]]
y=df['charges']
x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.4,random_state=5)
lm=LinearRegression()
lm.fit(x_train,y_train)
y_pred=lm.predict(x_test)
print(lm.coef_,lm.intercept_)
print(mean_squared_error(y_test,y_pred))

# print(df)
```

```
there is no null value
[ 2.73595953e+02 -1.72254657e+01  3.37942846e+02  1.86294065e+02
  2.47341011e+04  6.20560162e+02 -6.50266054e+02  6.60993216e+01] -12875.982149926318
37275584.81739203
```

In [70]:
```
# Consider variables x and y created from a pandas dataframe “car.csv” .
# Create new column named “Age_car” (Age_car=2023-year)
# For multiple linear regression problem, x contains the independent variables ( Age_car , Driven_kms , Fuel_Type , Sel
# variable which is to be predicted.Write a Python program to spilt x and y into training and testing datasets with a 2
# and print its coefficients ,intercept and mean squared error.
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
df=pd.read_csv('car.csv')
df['Age_car']=2023-df['Year']
df=df.drop(columns=['Car_Name'])
df=pd.get_dummies(data=df,drop_first=True)
df
x=df[['Age_car','Driven_kms','Fuel_Type_Petrol','Fuel_Type_Diesel','Selling_type_Individual','Transmission_Manual']]
y=df['Selling_Price']
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)
lm=LinearRegression()
lm.fit(x_train,y_train)
y_pred=lm.predict(x_test)
# print(y_pred)
print("coeffiecnets:",lm.coef_)
print("intercept:",lm.intercept_)
print("mean_squared_error:",mean_squared_error(y_test,y_pred))
```

```
coeffiecnets: [-4.03775557e-01 -4.86475839e-06  1.68430847e+00  6.21240413e+00
 -3.96019902e+00 -5.47824689e+00]
intercept: 12.131150273622477
mean_squared_error: 23.095669805657035
```

In [1]:

```
import SimpleGUICS2Pygame.simpleguics2pygame as simplegui
def draw_handler(canvas):
    canvas.draw_text('EXAMPLE OF TESTING',(50,190),10,'Black')
    canvas.draw_circle((50,50),20,1,'Red','Pink')
    canvas.draw_circle((150,50),20,1,'Green','Lime')
    canvas.draw_circle((50,140),20,1,'Green','Lime')
    canvas.draw_circle((150,140),20,1,'Red','Pink')
    canvas.draw_polygon([(75,75),(120,75),(120,120),(75,120)], 2, 'Blue','Aqua')
    canvas.draw_line((75,75),(50,50),1,'Black')
    canvas.draw_line((150,50),(120,75),1,'Black')
    canvas.draw_line((150,140),(120,120),1,'Black')
    canvas.draw_line((50,140),(75,120),1,'Black')

    frame.set_canvas_background('Yellow')

frame=simplegui.create_frame('testing',200,200)
frame.set_draw_handler(draw_handler)
frame.start()
```

In [6]:
```
# implementation of card game - Memory

import SimpleGUICS2Pygame.simpleguics2pygame as simplegui
import random

# helper function to initialize globals
def new_game():
    global deck, exposed, state, cIndex1, cIndex2, nScore, nMoves
    state, nScore, nMoves, cIndex1, cIndex2 = 0, 0, 0, -1, -1
    deck = [x for x in range(8)]*2
    random.shuffle(deck)
    exposed = [False]*16

# define event handlers
def mouseclick(pos):
    # add game state logic here
    global state, nScore, cIndex1, cIndex2, nMoves
    cardIndex = list(pos)[0]//50
    if not exposed[cardIndex]:
        if state == 0:                                          #just started
            cIndex1 = cardIndex
            exposed[cardIndex] = True
            state = 1
        elif state == 1:                                        #one card flipped
            cIndex2 = cardIndex
            exposed[cardIndex] = True
            if deck[cIndex1] == deck[cIndex2]:
                nScore += 1
            state = 2
            nMoves += 1
            label.set_text("Turns = " + str(nMoves))
        else:                                                   #two cards flipped
            if deck[cIndex1] != deck[cIndex2]:
                exposed[cIndex1], exposed[cIndex2] = False, False
                cIndex1, cIndex2 = -1, -1
            cIndex1 = cardIndex
            exposed[cardIndex] = True
            state = 1

# cards are logically 50x100 pixels in size
def draw(canvas):
```

```
    for i in range(16):
        if exposed[i]:
            canvas.draw_polygon([[i*50, 0], [(i+1)*50, 0], [(i+1)*50, 100], [i*50, 100]], 1, "Black", "White")
            canvas.draw_text(str(deck[i]), (i*50+11, 69), 55, "Black")
        else:
            canvas.draw_polygon([[i*50, 0], [(i+1)*50, 0], [(i+1)*50, 100], [i*50, 100]], 1, "Black", "Green")
    label.set_text("Turns = " + str(nMoves))


# create frame and add a button and labels
frame = simplegui.create_frame("Memory", 800, 100)
frame.add_button("Reset", new_game)
label = frame.add_label("Turns = 0")

# register event handlers
frame.set_mouseclick_handler(mouseclick)
frame.set_draw_handler(draw)

# get things rolling
new_game()
frame.start()
```

In [2]:

```
# Q-13
# Mouseclick Handlers
# Tic-Tac-Toe


# This program allows two users to play tic-tac-toe using
#   mouse input to the game.

import SimpleGUICS2Pygame.simpleguics2pygame as simplegui

# Global Variables

canvas_width = 300
canvas_height = 300
grid = [[" ", " ", " "], [" ", " ", " "], [" ", " ", " "]]
turn = "X"
won = False

# Helper Functions

def switch_turn():
    global turn, info
    if turn == "X":
        turn = "O"
    else:
        turn = "X"
    info.set_text("Player turn: " + turn)

# Returns 'True' if a player has won, false otherwise
def check_win():
    for a in range(0,3):
        if grid[a][0] != " " and grid[a][0] == grid[a][1] == grid[a][2]:
            return True
    for b in range(0,3):
        if grid[0][b] != " " and grid[0][b] == grid[1][b] == grid[2][b]:
            return True
    if grid[0][0] == grid[1][1] == grid[2][2] and grid[0][0] != " ":
        return True
    elif grid[0][2] == grid[1][1] == grid[2][0] and grid[0][2] != " ":
        return True
    else:
```

```
        return False

# Event Handlers

def draw(canvas):
    # Draws the grid lines
    canvas.draw_line([0, canvas_height // 3],
                     [canvas_width, canvas_height // 3], 1, "White")
    canvas.draw_line([0, canvas_height // 3 * 2],
                     [canvas_width, canvas_height // 3 * 2], 1, "White")
    canvas.draw_line([canvas_width // 3, 0],
                     [canvas_width // 3, canvas_height], 1, "White")
    canvas.draw_line([canvas_width // 3 * 2, 0],
                     [canvas_width // 3 * 2, canvas_height], 1, "White")

    # Draws the player choices using loops
    for r in range(0,3):
        for c in range(0,3):
            canvas.draw_text(grid[r][c],
                             [c * canvas_width // 3 + 20, r * canvas_height // 3 + 80], 80, "Red")

def click(pos):
    global won, info
    if not won:
        # Checks to see if the click was on a grid line
        if pos[0] % (canvas_width // 3) != 0 and pos[1] % (canvas_height // 3) != 0:
            r = pos[1] // (canvas_height // 3)
            c = pos[0] // (canvas_width // 3)
            # Checks to see if a square is already taken
            if grid[r][c] == " ":
                grid[r][c] = turn
                if check_win():
                    won = True
                    info.set_text("Player " + turn + " wins!")
                else:
                    switch_turn()

def reset():
    global grid, turn, won, info
    grid = [[" ", " ", " "], [" ", " ", " "], [" ", " ", " "]]
    turn = "X"
    won = False
```

```
    info.set_text("Player turn: " + turn)

# Frame

frame = simplegui.create_frame("Tic-Tac-Toe", canvas_width, canvas_height)

# Register Event Handlers

frame.set_draw_handler(draw)
frame.set_mouseclick_handler(click)
frame.add_button("Reset", reset)
info = frame.add_label("Player turn: " + turn)

# Start
frame.start()
```

In [2]:
```
# Q-14)
# Keyboard Input
# Shape Selection


# This is a simple program that allows you to change the
#   size, color, fill, and shape of an image on the screen.

import SimpleGUICS2Pygame.simpleguics2pygame as simplegui

# Global Variables

canvas_width = 200
canvas_height = 200
size = 100
colors = ["DeepPink", "Red", "DarkOrange", "Yellow", "Lime", "Green", "Blue",
          "Aqua", "Purple", "Magenta"]
fill = False
shapes = ["Square", "Circle", "Triangle"]
color_index = len(colors) // 2
shape_index = len(shapes) // 2

# Event Handlers

def draw(canvas):
    if shapes[shape_index] == "Square":
        x = canvas_width / 2 - size / 2
        y = canvas_height / 2 - size / 2
        if fill:
            canvas.draw_polygon([(x, y), (x + size, y), (x + size, y + size),
                                 (x, y + size)], 5,
                                colors[color_index], colors[color_index])
        else:
            canvas.draw_polygon([(x, y), (x + size, y), (x + size, y + size),
                                 (x, y + size)], 5,
                                colors[color_index])
    elif shapes[shape_index] == "Circle":
        if fill:
            canvas.draw_circle([canvas_width / 2, canvas_height / 2],
                               size / 2, 5,
                               colors[color_index], colors[color_index])
```

```
        else:
            canvas.draw_circle([canvas_width / 2, canvas_height / 2],
                               size / 2, 5,
                               colors[color_index])
    elif shapes[shape_index] == "Triangle":
        x = canvas_width / 2 - size / 2
        y = canvas_height / 2 - size / 2
        if fill:
            canvas.draw_polygon([(x, y + size), (x + size, y + size),
                                 (x + size / 2, y)], 5,
                                colors[color_index], colors[color_index])
        else:
            canvas.draw_polygon([(x, y + size), (x + size, y + size),
                                 (x + size / 2, y)], 5,
                                colors[color_index])

def key_handler(key):
    global fill, size, color_index, shape_index
    if key == simplegui.KEY_MAP['f']:
        fill = not fill
    elif key == simplegui.KEY_MAP['s']:
        if shape_index > 0:
            shape_index -= 1
    elif key == simplegui.KEY_MAP['d']:
        if shape_index < len(shapes) - 1:
            shape_index += 1
    elif key == simplegui.KEY_MAP['z']:
        if size > 10:
            size -= 10
    elif key == simplegui.KEY_MAP['x']:
        if size < 200:
            size += 10
    elif key == simplegui.KEY_MAP['c']:
        if color_index > 0:
            color_index -= 1
    elif key == simplegui.KEY_MAP['v']:
        if color_index < len(colors) - 1:
            color_index += 1

# Frame and Timer

frame = simplegui.create_frame("Shape Selection",
```

```
                                    canvas_width, canvas_height)
# This statement is necessary to declare a function to be the
#   keydown handler. Try changing the line that is commented
#   out to see the difference between a keyup handler and a
#   keydown handler.
frame.set_keydown_handler(key_handler)
#frame.set_keyup_handler(key_handler)
frame.add_label("Shape: s and d")
frame.add_label("Fill: f")
frame.add_label("Size: z and x")
frame.add_label("Color: c and v")

# Register Event Handlers

frame.set_draw_handler(draw)

# Start
frame.start()
```

In [1]:

```python
#Q-11
import SimpleGUICS2Pygame.simpleguics2pygame as simplegui
import math

# Initialize boolean variables for visibility of house parts
house_visible = False
roof_visible = False
window_visible = False

# Initialize colors
house_color = "Red"
roof_color = "Yellow"
window_color = "Blue"

# Function to draw the house
def draw_house(canvas):
    if house_visible:
        canvas.draw_polygon([(150, 300), (250, 300), (250, 200), (150, 200)], 1, house_color, house_color)
    if roof_visible:
        canvas.draw_polygon([(140, 200), (260, 200), (200, 140)], 1, roof_color, roof_color)
    if window_visible:
        canvas.draw_circle((200, 250), 15, 1, window_color, window_color)

# Handler for the "Reveal House" button
def reveal_house():
    global house_visible
    house_visible = not house_visible

# Handler for the "Reveal Roof" button
def reveal_roof():
    global roof_visible
    if house_visible:
        roof_visible = not roof_visible
    if (not house_visible and roof_visible):
        roof_visible = not roof_visible

# Handler for the "Reveal Window" button
def reveal_window():
    global window_visible
    if house_visible:
        window_visible = not window_visible
```

```
    if (not house_visible and window_visible):
        window_visible = not window_visible

# Create a frame and assign callbacks to buttons
frame = simplegui.create_frame("House Reveal Game", 400, 400)
frame.add_button("Reveal House (Red)", reveal_house)
frame.add_button("Reveal Roof (Yellow)", reveal_roof)
frame.add_button("Reveal Window (Blue)", reveal_window)
frame.set_draw_handler(draw_house)

# Start the frame
frame.start()
```

In [4]:

```python
#Q-12
import SimpleGUICS2Pygame.simpleguics2pygame as simplegui
import random
import math

# Initialize the canvas size
CANVAS_WIDTH = 400
CANVAS_HEIGHT = 400

# Initialize the lists to hold shape objects
circles = []
triangles = []
squares = []

# Function to create a random color
def random_color():
    return "rgb(" + str(random.randint(0, 255)) + "," + str(random.randint(0, 255)) + "," + str(random.randint(0, 255))

# Function to create a random position within the canvas boundaries
def random_position():
    x = random.randint(0, CANVAS_WIDTH)
    y = random.randint(0, CANVAS_HEIGHT)
    return (x, y)

# Function to draw shapes on the canvas
def draw(canvas):
    for circle in circles:
        canvas.draw_circle(circle[0], 20, 2, random_color(), random_color())
    for triangle in triangles:
        triangle_points = [(triangle[0][0], triangle[0][1] + 20), (triangle[0][0] - 20, triangle[0][1] - 20), (triangle
        canvas.draw_polygon(triangle_points, 2, random_color(), random_color())
    for square in squares:
        square_points = [(square[0][0] - 20, square[0][1] - 20), (square[0][0] - 20, square[0][1] + 20), (square[0][0]
        canvas.draw_polygon(square_points, 2, random_color(), random_color())

# Function to create or remove circles
def toggle_circles():
    global circles
    if not circles:
        circles = [(random_position(), random_color()) for _ in range(10)]
    else:
```

```
        circles = []

# Function to create or remove triangles
def toggle_triangles():
    global triangles
    if not triangles:
        triangles = [(random_position(), random_color()) for _ in range(10)]
    else:
        triangles = []

# Function to create or remove squares
def toggle_squares():
    global squares
    if not squares:
        squares = [(random_position(), random_color()) for _ in range(10)]
    else:
        squares = []

# Create a frame and assign callbacks to buttons
frame = simplegui.create_frame("Shapes on Canvas", CANVAS_WIDTH, CANVAS_HEIGHT)
frame.set_draw_handler(draw)
frame.add_button("Circles", toggle_circles)
frame.add_button("Triangles", toggle_triangles)
frame.add_button("Squares", toggle_squares)

# Start the frame
frame.start()
```

In [ ]: