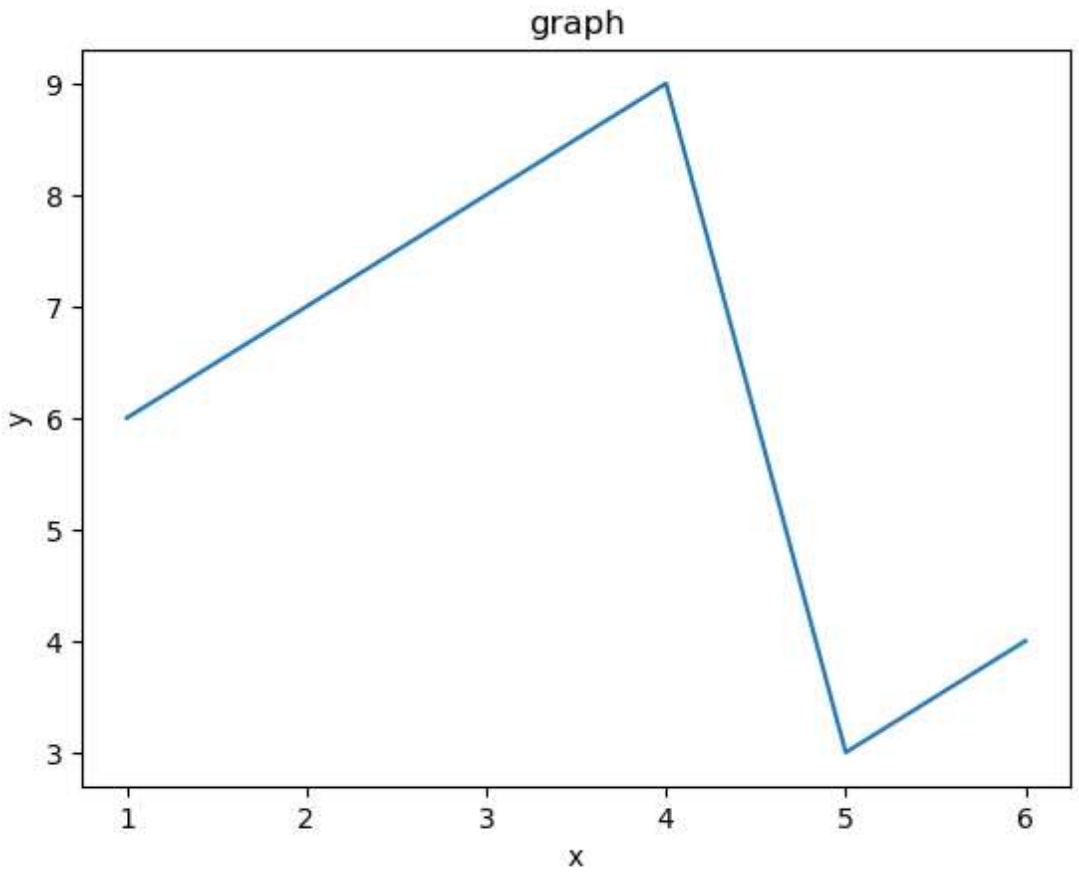
In [2]:
```
#1)
import numpy as np
a=np.array([[1,2,3,],[4,5,6]])
print('array type is :',type(a))
print("shape of array:",a.shape)
print('size of array:',a.size)
print('array elements type :',a.dtype)
print('dimension of array:',a.ndim)
print(a[1,...])
print('\n')
print(a[...,1])
print('\n')
print(a[...,1:])
print('\n')
print(a[1:])
```

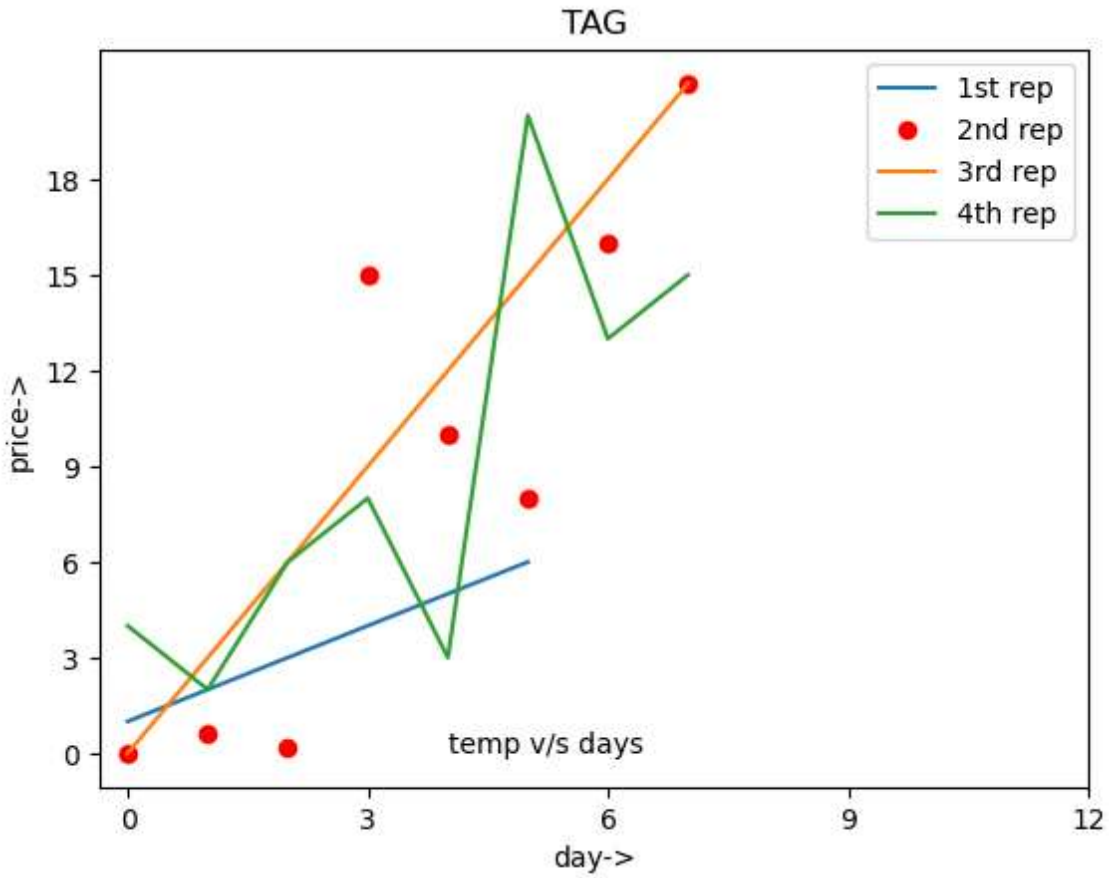
```
array type is : <class 'numpy.ndarray'>
shape of array: (2, 3)
size of array: 6
array elements type : int32
dimension of array: 2
[4 5 6]


[2 5]


[[2 3]
 [5 6]]


[[4 5 6]]
```

In [24]:
```
#2)
import pandas as pd
import numpy as np
data=[1,2,3,4,5,6]
df=pd.DataFrame(data,columns=['numbers'])
print(df,'\n')
data1=[[1,'kumar'],[2,'raam'],[3,'soma']]
sp=pd.DataFrame(data1,columns=['id','names'])
print(sp,'\n')
data2={
    'name': ['kumar','raam','bob'],
    'age': [23,24,21],
    'city': ['newyork','bangalore','bhadra']
}
rt=pd.DataFrame(data2)
print(rt,'\n')
my_df = pd.DataFrame(data=[4,5,6,7], index=range(0,4), columns=['A'])
print(pd.DataFrame(my_df))
print('\n')
my_dict = {1: ['1', '3'], 2: ['1', '2'], 3: ['2', '4']}
print(pd.DataFrame(my_dict))
```

```
   numbers
0        1
1        2
2        3
3        4
4        5
5        6

   id  names
0   1  kumar
1   2   raam
2   3   soma

    name  age       city
0  kumar   23    newyork
1   raam   24  bangalore
2    bob   21     bhadra

   A
0  4
1  5
2  6
3  7


   1  2  3
0  1  1  2
1  3  2  4
```

In [27]:
```
#3)
import matplotlib.pyplot as plt
x=[1,2,3,4,5,6]
y=[6,7,8,9,3,4]
plt.plot(x,y)
plt.xlabel('x')
plt.ylabel('y')
plt.title('graph')
plt.show()

a=[1,2,3,4,5,6]
b=[0,0.6,0.2,15,10,8,16,21]
plt.plot(a)
plt.plot(b,"or")
plt.plot(list(range(0,22,3)))
plt.xlabel("day->")
plt.ylabel("price->")
plt.title("TAG")
c=[4, 2, 6, 8, 3, 20, 13, 15]
plt.plot(c,label="4th rep")
plt.xticks(list(range(0,15,3)))
plt.yticks(list(range(0,20,3)))
plt.legend(['1st rep','2nd rep','3rd rep','4th rep'])
plt.annotate("temp v/s days",xy=(4,0))
plt.show()
```




TAG
1st rep
2nd rep
3rd rep
4th rep
price->
day->
temp v/s days
0
3
6
9
12
18
15

In [31]:
```
#4)
import numpy as np
vector_a=np.array([1,2,3])
vector_b=np.array([4,5,6])
addition=np.add(vector_a,vector_b)
subtraction=np.subtract(vector_a,vector_b)
multiply=np.dot(vector_b,vector_a)
mean_a=np.mean(vector_a)
magnitude=np.linalg.norm(vector_a)
distance=np.linalg.norm(vector_a-vector_b)
sumofsquares=np.sum(np.square(vector_a))
print(addition,'\n')
print(subtraction,'\n')
print(multiply,'\n')
print(mean_a,'\n')
print(magnitude,'\n')
print(distance,'\n')
print(sumofsquares)
```

```
[5 7 9]

[-3 -3 -3]

32

2.0

3.7416573867739413

5.196152422706632

14
```

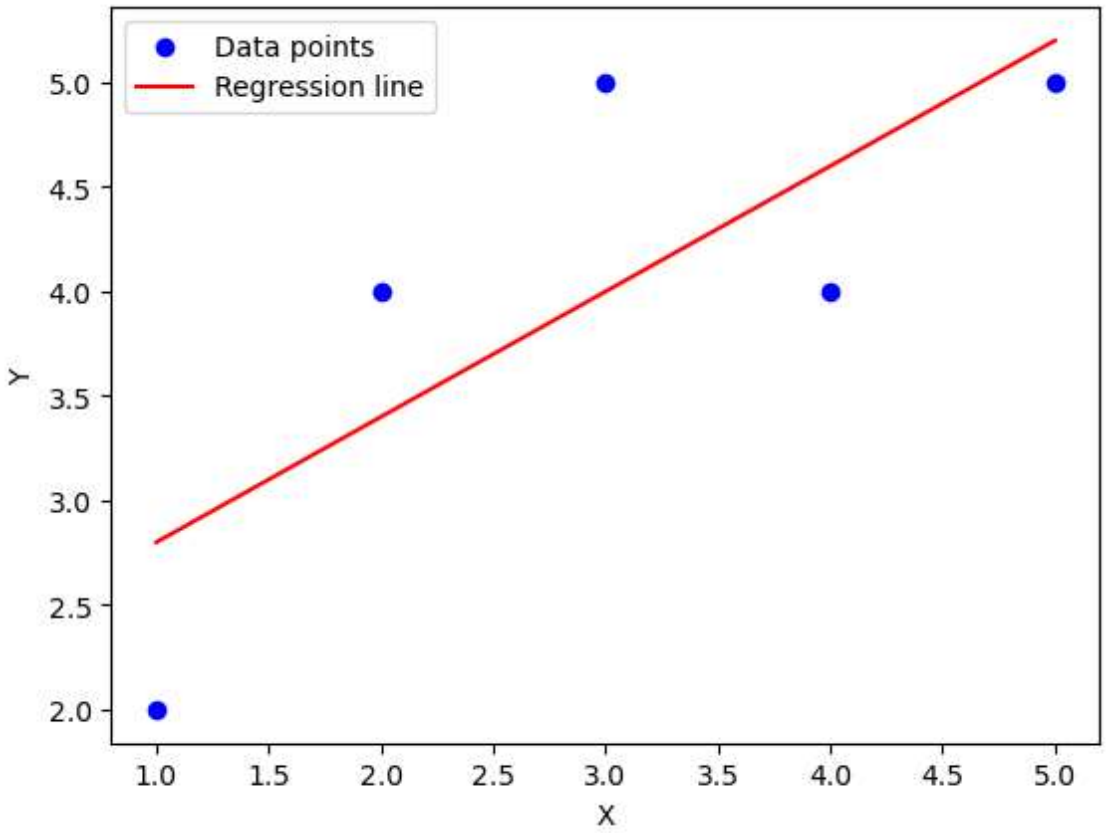
In [34]:
```
#5)
import pandas as pd
data={
    'x':[1,2,3,4,5],
    'y':[2,2,3,4,5]
}
df=pd.DataFrame(data)
covriance_matrix=df.cov()
covariance=covriance_matrix.loc['x','y']
correlation_matrix=df.corr()
correlation=correlation_matrix.loc['x','y']
print(f" covariance:{covariance}")
print(f" correlation:{correlation}")
```

```
 covariance:2.0
 correlation:0.9701425001453321
```

In [35]:
```
#7
def gd(sp,lp,ni):
    def sq(x):
        return x**2
    x=sp
    for i in range(ni):
        grad=2*x
        x=x-lp*grad
        print(f"iteration{i+1}:x={x},f(x)={x**2}")
    return x
sp=10
lp=0.1
ni=10
x_min=gd(sp,lp,ni)
print(f"optimize at x={x_min},f(x)={x_min**2}")
print(f'\nAfter {ni} iterations, the optimized value is x = {x_min}')
```

```
iteration1:x=8.0,f(x)=64.0
iteration2:x=6.4,f(x)=40.96000000000001
iteration3:x=5.12,f(x)=26.2144
iteration4:x=4.096,f(x)=16.777216
iteration5:x=3.2768,f(x)=10.73741824
iteration6:x=2.62144,f(x)=6.871947673600001
iteration7:x=2.0971520000000003,f(x)=4.398046511104002
iteration8:x=1.6777216000000004,f(x)=2.8147497671065613
iteration9:x=1.3421772800000003,f(x)=1.801439850948199
iteration10:x=1.0737418240000003,f(x)=1.1529215046068475
optimize at x=1.0737418240000003,f(x)=1.1529215046068475

After 10 iterations, the optimized value is x = 1.0737418240000003
```

In [40]:
```
#8
import numpy as np
import matplotlib.pyplot as plt

def simple_linear_regression(x, y):
    x_mean = np.mean(x)
    y_mean = np.mean(y)

    b1 = np.sum((x - x_mean) * (y - y_mean)) / np.sum((x - x_mean)**2)
    b0 = y_mean - b1 * x_mean

    return b0, b1
x = np.array([1, 2, 3, 4, 5])
y = np.array([2, 4, 5, 4, 5])
b0, b1 = simple_linear_regression(x, y)
y_pred = b0 + b1 * x
plt.scatter(x, y, color='blue', label='Data points')
plt.plot(x, y_pred, color='red', label='Regression line')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.show()
print(f"Linear Regression Line: y = {b0:.2f} + {b1:.2f}x")
```




```
Linear Regression Line: y = 2.20 + 0.60x
```

In [39]:
```python
#9

def accuracy(tp, fp, fn, tn):
    correct = tp + tn
    total = tp + fp + fn + tn
    return correct / total

def precision(tp, fp, fn, tn):
    return tp / (tp + fp)

def recall(tp, fp, fn, tn):
    return tp / (tp + fn)

def f1_score(tp, fp, fn, tn):
    p = precision(tp, fp, fn, tn)
    r = recall(tp, fp, fn, tn)

    return 2 * p * r / (p + r)

def evaluate_classification(tp, fp, fn, tn):
    acc = accuracy(tp, fp, fn, tn)
    prec = precision(tp, fp, fn, tn)
    rec = recall(tp, fp, fn, tn)
    f1 = f1_score(tp, fp, fn, tn)

    print(f"Accuracy: {acc:.2f}")
    print(f"Precision: {prec:.2f}")
    print(f"Recall: {rec:.2f}")
    print(f"F1 Score: {f1:.2f}")

# Take user input for the confusion matrix values
tp = int(input("Enter True Positive (tp): "))
fp = int(input("Enter False Positive (fp): "))
fn = int(input("Enter False Negative (fn): "))
tn = int(input("Enter True Negative (tn): "))

# Evaluate the classification performance
evaluate_classification(tp, fp, fn, tn)
```

```
Enter True Positive (tp): 6
Enter False Positive (fp): 9
Enter False Negative (fn): 3
Enter True Negative (tn): 8
Accuracy: 0.54
Precision: 0.40
Recall: 0.67
F1 Score: 0.50
```

In [41]:
```
import random
def random_kid():
    return random.choice(["boy","girl"])

both_girls=0
either_girl=0
older_girl=0

for _ in range(1000):
    younger=random_kid()
    older=random_kid()

    if younger=="girl" and older=="girl":
        both_girls+=1
    if younger=="girl" or older=="girl":
        either_girl+=1
    if older=="girl":
        older_girl+=1

print("the probability of both girls where older is a girl:",both_girls/older_gir
print("the probability of both girls where one is a girl:",both_girls/either_girl
```

```
the probability of both girls where older is a girl: 0.4843137254901961
the probability of both girls where one is a girl: 0.3302139037433155
```

In [ ]:
```

```