In [1]:
```
#task1
import numpy as np
a=np.random.random((10,3))
new_a=(np.abs(a-0.5))
otv=new_a.min(axis=1)
print("new_a:\n", new_a)
print("otv:",otv)
```

```
new_a:
 [[0.19346339 0.12969041 0.19900313]
 [0.30197644 0.08375684 0.25014704]
 [0.37391278 0.07560692 0.11955331]
 [0.1135419  0.12406444 0.46682893]
 [0.21556978 0.1319648  0.03536751]
 [0.37156881 0.48526903 0.24297718]
 [0.49989439 0.31799655 0.01718092]
 [0.3029102  0.07134247 0.32796297]
 [0.37147994 0.43330618 0.01590673]
 [0.08169281 0.47508833 0.36555951]]
otv: [0.12969041 0.08375684 0.07560692 0.1135419  0.03536751 0.24297
718
 0.01718092 0.07134247 0.01590673 0.08169281]
```

In [3]:
```
#task2
import numpy as np
a=np.random.random((6,6))
sum_po_stroke=np.sum(a,axis=1)
min_v_stolbike=np.min(a,axis=0)
otv=sum_po_stroke/min_v_stolbike
print("a=:\n",a)
print("sum_po_stroke=",sum_po_stroke)
print("min_v_stolbike:",min_v_stolbike)
print("otv=",otv)
```

```
a=:
 [[0.6949285  0.31889747 0.69187686 0.69021296 0.68739329 0.82224288
]
 [0.61584352 0.73630933 0.29682983 0.16715735 0.03357313 0.10777971]
 [0.15877933 0.95818089 0.51756565 0.7049404  0.86184375 0.78103447]
 [0.00291541 0.34651774 0.66741407 0.31393654 0.95394828 0.72876263]
 [0.04925694 0.7647517  0.23764398 0.54285374 0.34854382 0.57927978]
 [0.36005636 0.04512146 0.20820883 0.20025033 0.8445041  0.06440261]
]
sum_po_stroke= [3.90555196 1.95749286 3.9823445  3.01349467 2.522329
97 1.72254369]
min_v_stolbike: [0.00291541 0.04512146 0.20820883 0.16715735 0.03357
313 0.06440261]
otv= [1339.6241039    43.38274809   19.12668443   18.02789229   75.1
2942489
   26.7464883 ]
```

In [4]:
```
#task3
import numpy as np
a=np.array([6,2,0,3,0,0,5,7,0])
mask= (a==0)
new_a=a[1:len(a):1] #cut off the first
new_mask=mask[0:-1:1] #cut off the last
fitting=new_a[new_mask]
print("a=:",a)
print("mask=:",mask)
print("new_a=:",new_a)
print("new_mask=:",new_mask)
print("fitting:", fitting)
print(np.max(fitting))
```

```
a=: [6 2 0 3 0 0 5 7 0]
mask=: [False False  True False  True  True False False  True]
new_a=: [2 0 3 0 0 5 7 0]
new_mask=: [False False  True False  True  True False False]
fitting: [3 0 5]
5
```

In [5]:
```
#task4
import numpy as np
x=np.ones(10)
i=np.array([0,1,2,3,5,5,5,8])
print("x=:",x)
print("i=:",i)
np.add.at(x,i,1)
print("otv=:",x)
```

```
x=: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
i=: [0 1 2 3 5 5 5 8]
otv=: [2. 2. 2. 2. 1. 4. 1. 1. 2. 1.]
```

In [6]:
```
#task5
import numpy as np
a=np.arange(16).reshape(4,4)
print("a=:\n",a)
d=dict()
for i in range(0,4-1,1):
    tek_stroka=i
    tek_diag=[]
    for j in range(0,tek_stroka+1,1):
        tek_diag.append(a[tek_stroka,j])
        tek_stroka=tek_stroka-1
    d.update({i:tek_diag})

for j in range(1,4,1):
    tek_stolb=j
    tek_diag=[]
    for i in range(4-1,j-1,-1):
        tek_diag.append(a[i,tek_stolb])
        tek_stolb=tek_stolb+1
    d.update({(4-1+j):tek_diag})
print(d)
```

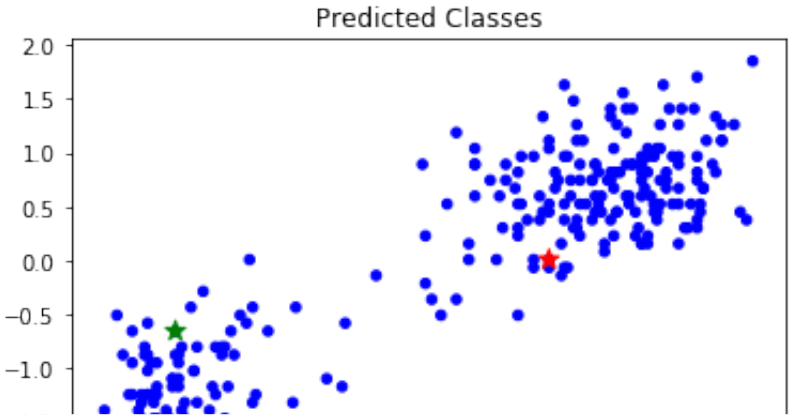
```
a=:
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]
{0: [0], 1: [4, 1], 2: [8, 5, 2], 4: [13, 10, 7], 5: [14, 11], 6: [1
5]}
```

In [7]:
```
#task6: kmeans algoritm
import numpy as np
import matplotlib.pyplot as plt

def make_graphic(data):
    plt.scatter(*data.T)
    plt.xlabel('Eruption time (min)')
    plt.ylabel('Waiting time til next eruption (min)')
    plt.show()

def make_graphic_classes(data,labels):
    plt.scatter(*data.T, c=np.where(labels, "blue", "orange"), s=20)
    plt.scatter(*centroids.T, c=["red", "green"], s=95, marker='*')
    plt.title('Predicted Classes')
    plt.show()

def mark_distances_old(data,centroids):
    raznost1=(data-centroids[0])
    raznost2=(data-centroids[1])
    distance=[]
```

```
    for i in range(0,len(raznost1),1):
        b=[np.linalg.norm(raznost1[i]),np.linalg.norm(raznost2[i])]
        distance.append(b)
    return distance

def mark_distances(data,centroids):
    raznost1=(data-centroids[0])
    raznost2=(data-centroids[1])

    distTo1=np.linalg.norm(raznost1,axis=1)
    distTo2=np.linalg.norm(raznost2,axis=1)
    distance = np.hstack([distTo1.reshape(272,1),distTo2.reshape(272,1
    return distance

def get_label(data,centroids):

    dist=mark_distances(data,centroids)
    label=np.argmin(dist, axis=1)

    return label

def one_step(data,centroids,labels):
    #make_graphic_classes(data,labels)
    labels=get_label(data,centroids)
    #print("labels: ",labels[0],labels[1],labels[2])
    #print(labels)
    #make_graphic_classes(data,labels)


    mask=(labels==0)
    center0=data[mask].mean(axis=0)
    mask=(labels==1)
    center1=data[mask].mean(axis=0)
    print("center0, center1: " ,center0,center1)

    delta0=np.linalg.norm(center0-centroids[0])
    delta1=np.linalg.norm(center1-centroids[1])

    centroids[0]=center0
    centroids[1]=center1


    return centroids,labels,delta0,delta1
```

In [9]:
```
data = np.loadtxt('http://www.stat.cmu.edu/~larry/all-of-statistics/=d
#make_graphic(data,labels)

data=data-data.mean(axis=0);
data=data/data.std(axis=0)
print("data: ",data[0],data[1],data[2])

#centroids = np.random.uniform(-2, 2, 4).reshape((2, 2))
centroids=data[73:75:1]
print("centroids: ",centroids)

delta0=1
delta1=1
global labels
labels=np.ones(272)

print("start process:\n")

i=1

while(max(delta0,delta1)>0.000001):
    print("step=",i)
    make_graphic_classes(data,labels)
    i=i+1
    centroids,labels,delta0,delta1=one_step(data,centroids,labels)
    print("delta0, delta1 : ",delta0,delta1)

make_graphic_classes(data,labels)
```

```
data:  [0.09849886 0.59712344] [-1.48145856 -1.24518118] [-0.1358614
9  0.22866251]
centroids:  [[ 0.44960051  0.00758596]
 [-1.32082956 -0.6556437 ]]
start process:

step= 1
```



In [ ]: