Part 1: V-Learning

```
In [1]: import math
        import matplotlib.pyplot as plt
        import numpy as np
        plt.rcParams["figure.figsize"] = (20,10)
```

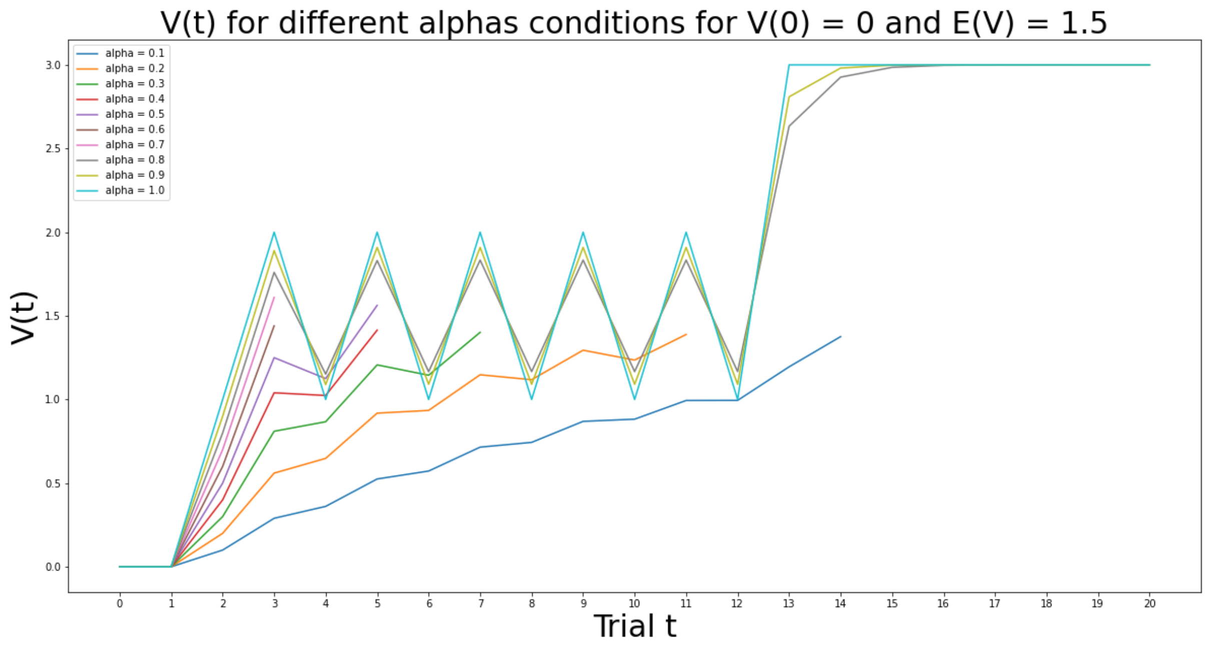
Play with initial condition : V(0)

a) V(0) = 0

```
In [21]: import statistics
         R=[0,1,2,1,2,1,2,1,2,1,2,1,3,3,3,3,3,3,3,3]
         statistics.median(R[0:12])
```

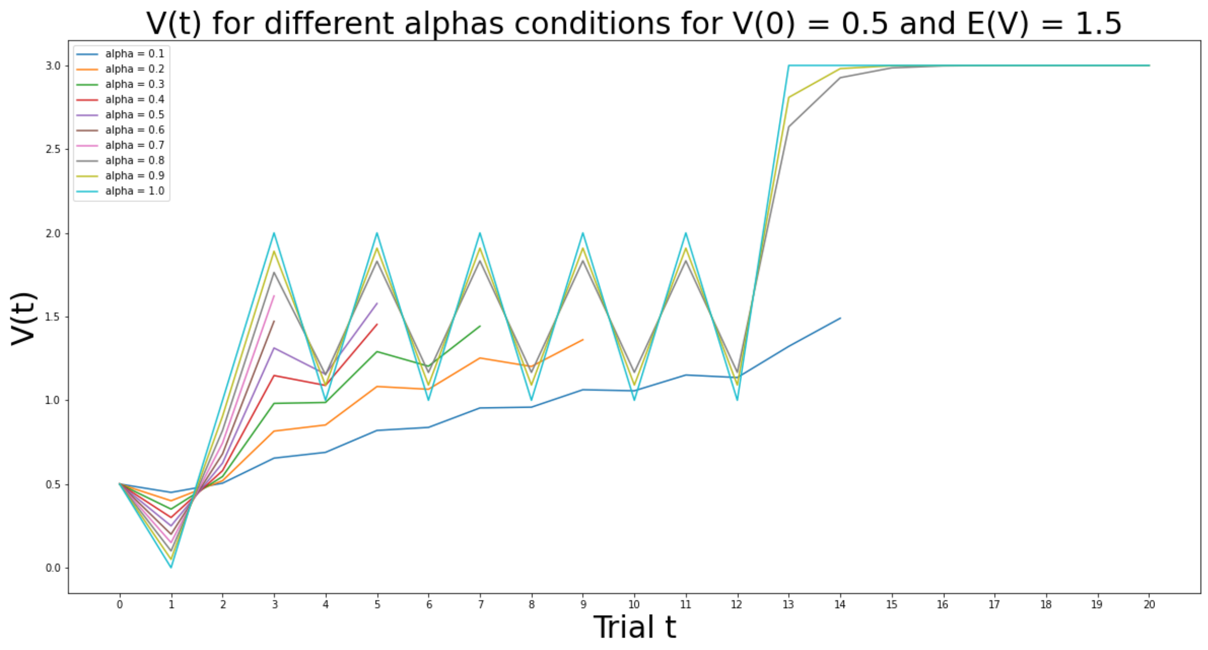
```
Out[21]: 1.0
```

In [2]:
```python
alphas = np.linspace(0.1,1,10)
V_alphas = []
initial_condition = 0
R = [0,0,1,2,1,2,1,2,1,2,1,2,1,3,3,3,3,3,3,3,3]  # add 0 at the beginnin
g (not used just for indexation purposes)
E_v = 1.5
for alpha in alphas:
    V=[initial_condition]
    for  t in range(1,len(R)):

        if(abs(V[t-1]-E_v)>0.1*E_v):
            V.append(V[t-1] + alpha * (R[t]-V[t-1]))
        else :
            break
    V_alphas.append(V)

for i in range(len(alphas)):
    plt.plot(V_alphas[i],label='alpha = '+str(round(alphas[i],2)))
    plt.xlabel("Trial t",fontsize=30)
    plt.ylabel("V(t)",fontsize=30)
    plt.xticks(range(0,21))
    plt.title ("V(t) for different alphas conditions for V(0) = "+ str(i
nitial_condition)+ " and E(V) = " +str(E_v),fontsize=30)
    plt.legend()
```



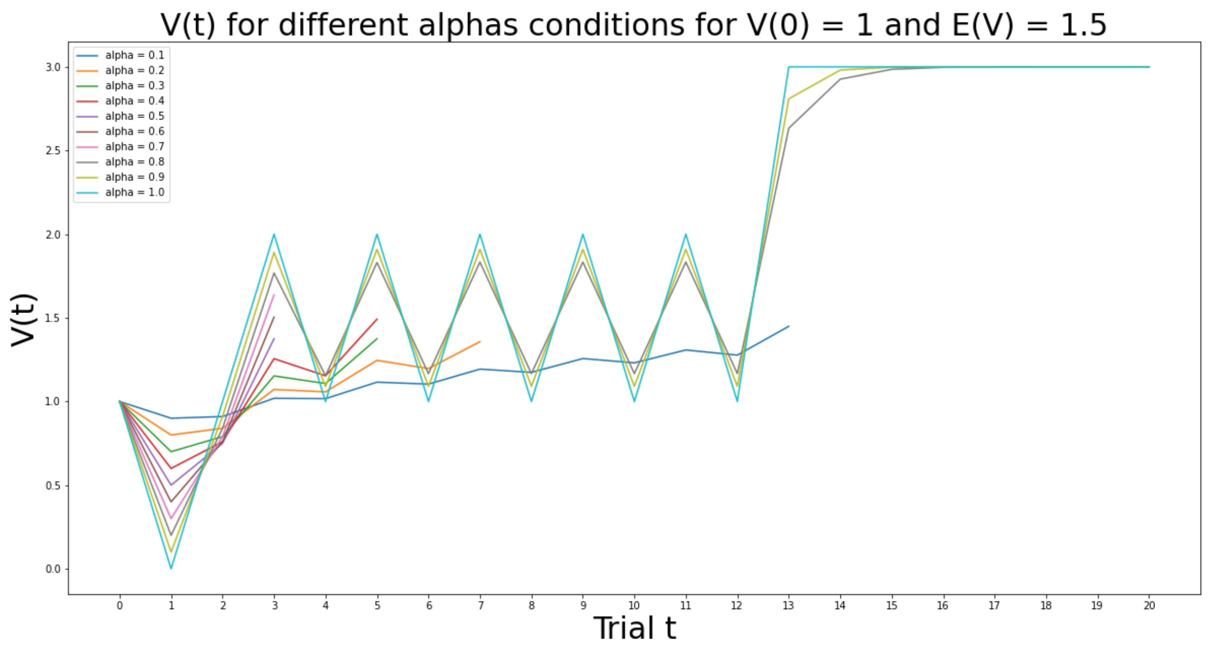
b) V(0) = 0.5

In [213]:
```
alphas = np.linspace(0.1,1,10)
V_alphas = []
initial_condition = 0.5
R = [0,0,1,2,1,2,1,2,1,2,1,2,1,3,3,3,3,3,3,3,3]  # add 0 at the beginnin
g
E_v = 1.5
for alpha in alphas:
    V=[initial_condition]
    for  t in range(1,len(R)):

        if(abs(V[t-1]-E_v)>0.1*E_v):
            V.append(V[t-1] + alpha * (R[t]-V[t-1]))
        else :
            break
    V_alphas.append(V)

for i in range(len(alphas)):
    plt.plot(V_alphas[i],label='alpha = '+str(round(alphas[i],2)))
    plt.xlabel("Trial t",fontsize=30)
    plt.ylabel("V(t)",fontsize=30)
    plt.xticks(range(0,21))
    plt.title ("V(t) for different alphas conditions for V(0) = "+ str(i
nitial_condition)+ " and E(V) = " +str(E_v),fontsize=30)
    plt.legend()
```



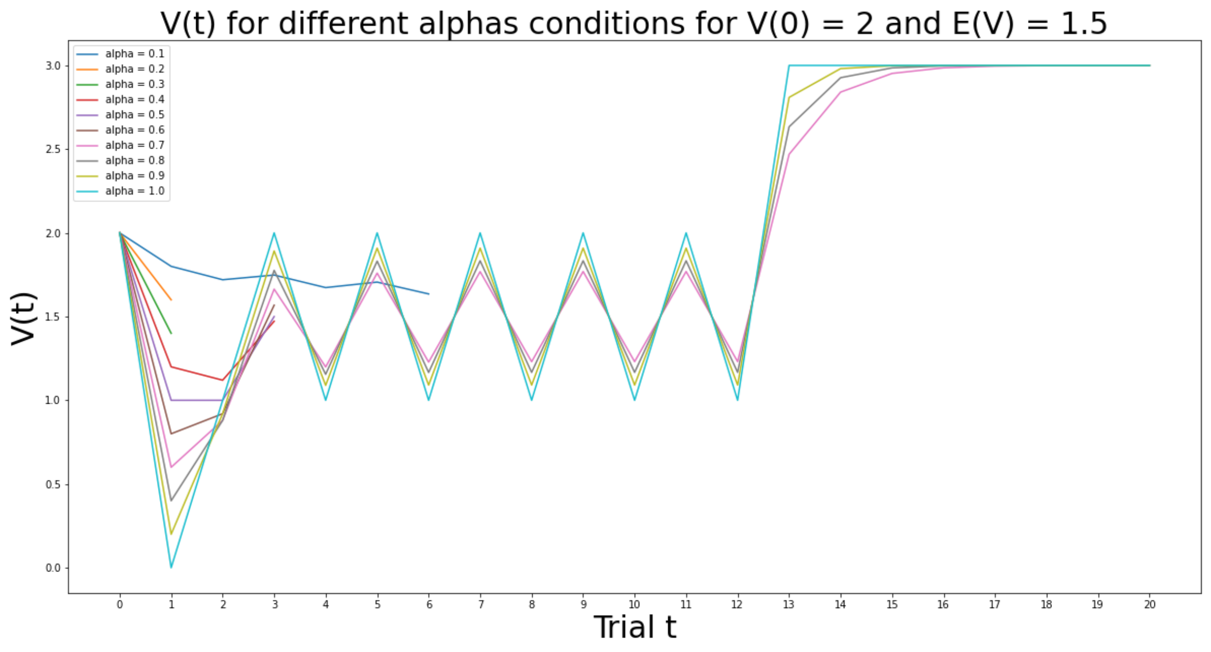
b) V(0) = 1

In [214]:
```
alphas = np.linspace(0.1,1,10)
V_alphas = []
initial_condition = 1
R = [0,0,1,2,1,2,1,2,1,2,1,2,1,3,3,3,3,3,3,3,3]  # add 0 at the beginnin
g
E_v = 1.5
for alpha in alphas:
    V=[initial_condition]
    for  t in range(1,len(R)):

        if(abs(V[t-1]-E_v)>0.1*E_v):
            V.append(V[t-1] + alpha * (R[t]-V[t-1]))
        else :
            break
    V_alphas.append(V)

for i in range(len(alphas)):
    plt.plot(V_alphas[i],label='alpha = '+str(round(alphas[i],2)))
    plt.xlabel("Trial t",fontsize=30)
    plt.ylabel("V(t)",fontsize=30)
    plt.xticks(range(0,21))
    plt.title ("V(t) for different alphas conditions for V(0) = "+ str(i
nitial_condition)+ " and E(V) = " +str(E_v),fontsize=30)
    plt.legend()
```



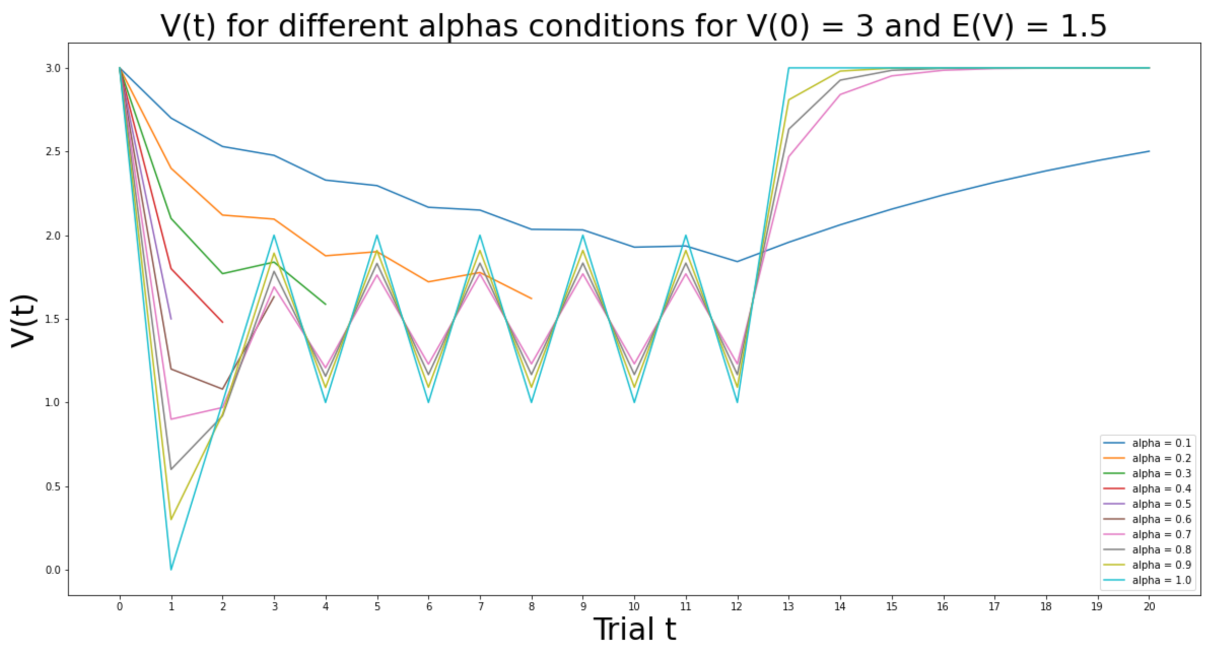
b) V(0) = 2

In [215]:
```
alphas = np.linspace(0.1,1,10)
V_alphas = []
initial_condition = 2
R = [0,0,1,2,1,2,1,2,1,2,1,2,1,3,3,3,3,3,3,3,3]  # add 0 at the beginnin
g
E_v = 1.5
for alpha in alphas:
    V=[initial_condition]
    for  t in range(1,len(R)):

        if(abs(V[t-1]-E_v)>0.1*E_v):
            V.append(V[t-1] + alpha * (R[t]-V[t-1]))
        else :
            break
    V_alphas.append(V)

for i in range(len(alphas)):
    plt.plot(V_alphas[i],label='alpha = '+str(round(alphas[i],2)))
    plt.xlabel("Trial t",fontsize=30)
    plt.ylabel("V(t)",fontsize=30)
    plt.xticks(range(0,21))
    plt.title ("V(t) for different alphas conditions for V(0) = "+ str(i
nitial_condition)+ " and E(V) = " +str(E_v),fontsize=30)
    plt.legend()
```



b) V(0) = 3

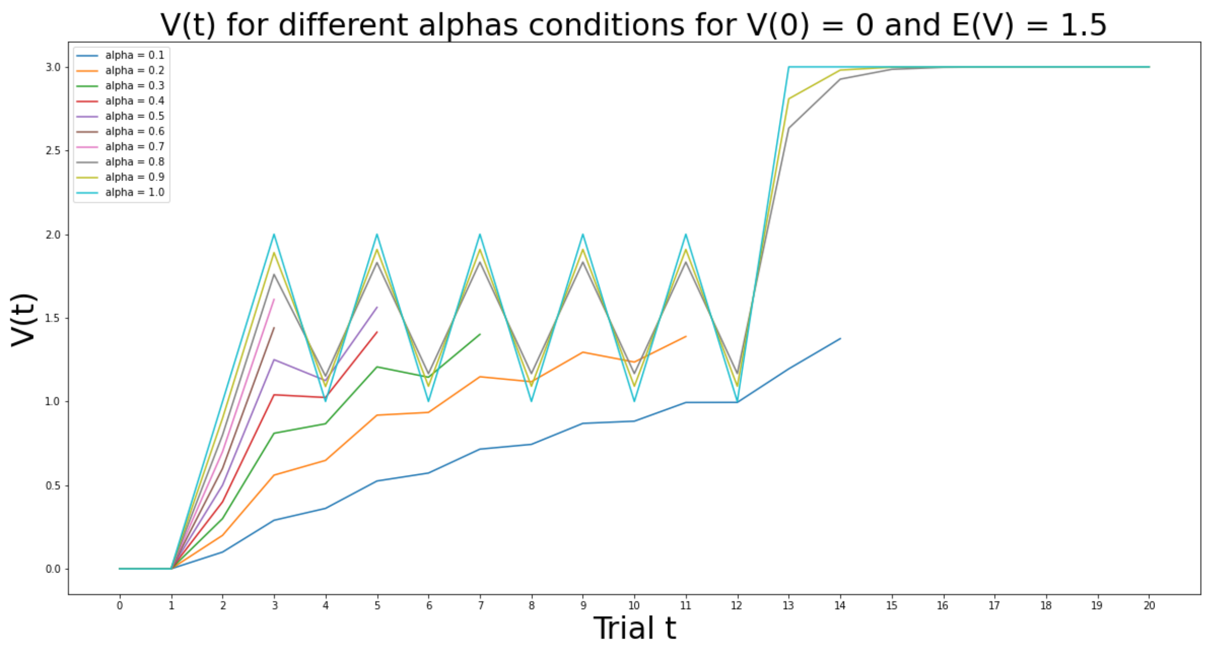
In [216]:
```
alphas = np.linspace(0.1,1,10)
V_alphas = []
initial_condition = 3
R = [0,0,1,2,1,2,1,2,1,2,1,2,1,3,3,3,3,3,3,3,3]  # add 0 at the beginnin
g
E_v = 1.5
for alpha in alphas:
    V=[initial_condition]
    for  t in range(1,len(R)):

        if(abs(V[t-1]-E_v)>0.1*E_v):
            V.append(V[t-1] + alpha * (R[t]-V[t-1]))
        else :
            break
    V_alphas.append(V)

for i in range(len(alphas)):
    plt.plot(V_alphas[i],label='alpha = '+str(round(alphas[i],2)))
    plt.xlabel("Trial t",fontsize=30)
    plt.ylabel("V(t)",fontsize=30)
    plt.xticks(range(0,21))
    plt.title ("V(t) for different alphas conditions for V(0) = "+ str(i
nitial_condition)+ " and E(V) = " +str(E_v),fontsize=30)
    plt.legend()
```



Let's now use V(0)=0 for initial condition to respond to the questions :

(a) E(V)=1.5

In [22]:
```
alphas = np.linspace(0.1,1,10)
V_alphas = []
initial_condition = 0
R = [0,0,1,2,1,2,1,2,1,2,1,2,1,3,3,3,3,3,3,3,3]  # add 0 at the beginnin
g (not used just for  indexation purposes)
E_v = 1.5
for alpha in alphas:
    V=[initial_condition]
    for  t in range(1,len(R)):

        if(abs(V[t-1]-E_v)>0.1*E_v):
            V.append(V[t-1] + alpha * (R[t]-V[t-1]))
        else :
            break
    V_alphas.append(V)

for i in range(len(alphas)):
    plt.plot(V_alphas[i],label='alpha = '+str(round(alphas[i],2)))
    plt.xlabel("Trial t",fontsize=30)
    plt.ylabel("V(t)",fontsize=30)
    plt.xticks(range(0,21))
    plt.title ("V(t) for different alphas conditions for V(0) = "+ str(i
nitial_condition)+ " and E(V) = " +str(E_v),fontsize=30)
    plt.legend()
```



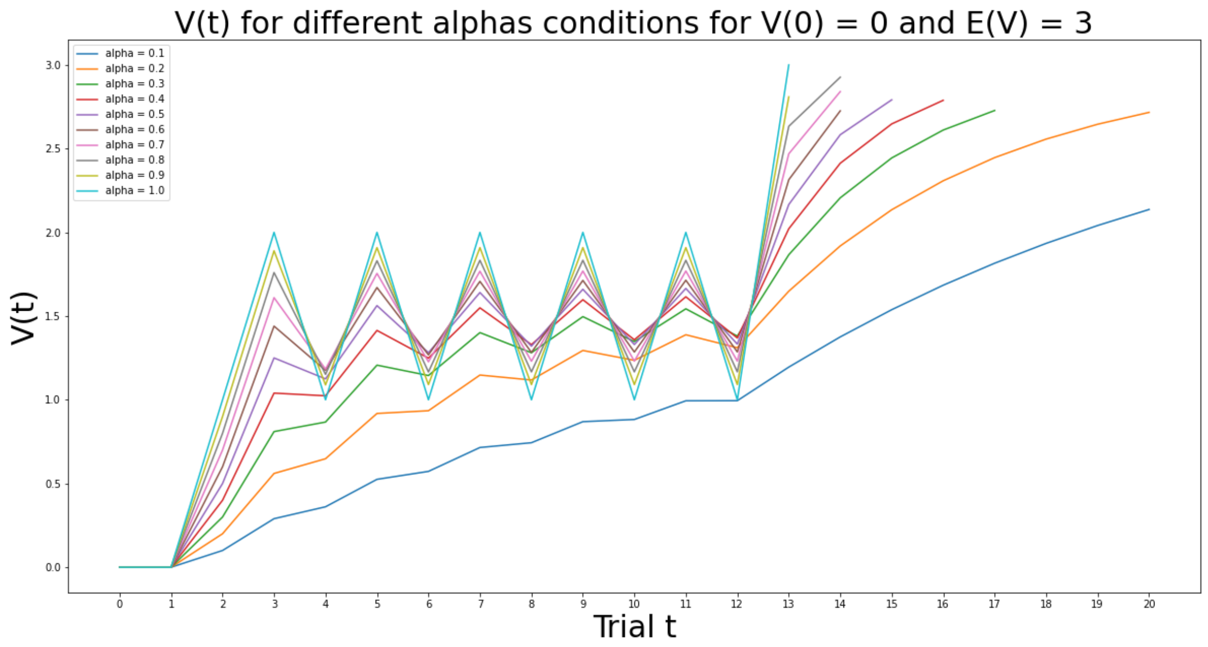
In [23]:
```python
for i in range(len(alphas)):
    if (abs(V_alphas[i][-1] - E_v)<=0.1*E_v):
        print ("For alpha = " + str(round(alphas[i],2)) + " , it converg
es after : " +str(len(V_alphas[i])-1) +" trials.")
```

```
For alpha = 0.1 , it converges after : 14 trials.
For alpha = 0.2 , it converges after : 11 trials.
For alpha = 0.3 , it converges after : 7 trials.
For alpha = 0.4 , it converges after : 5 trials.
For alpha = 0.5 , it converges after : 5 trials.
For alpha = 0.6 , it converges after : 3 trials.
For alpha = 0.7 , it converges after : 3 trials.
```

alpha=0.6 and 0.7 converge the quickest for E(V)=1.5. I think it makes sense

(b) E(V)=3

In [234]:
```
alphas = np.linspace(0.1,1,10)
V_alphas = []
initial_condition = 0
R = [0,0,1,2,1,2,1,2,1,2,1,2,1,3,3,3,3,3,3,3,3]  # add 0 at the beginnin
g
E_v = 3
for alpha in alphas:
    V=[initial_condition]
    for  t in range(1,len(R)):

        if(abs(V[t-1]-E_v)>0.1*E_v):
            V.append(V[t-1] + alpha * (R[t]-V[t-1]))
        else :
            break
    V_alphas.append(V)

for i in range(len(alphas)):
    plt.plot(V_alphas[i],label='alpha = '+str(round(alphas[i],2)))
    plt.xlabel("Trial t",fontsize=30)
    plt.ylabel("V(t)",fontsize=30)
    plt.xticks(range(0,21))
    plt.title ("V(t) for different alphas conditions for V(0) = "+ str(i
nitial_condition)+ " and E(V) = " +str(E_v),fontsize=30)
    plt.legend()
```



In [235]:
```
for i in range(len(alphas)):
    if (abs(V_alphas[i][-1] - E_v)<=0.1*E_v):
        print ("For alpha = " + str(round(alphas[i],2)) + " , it converg
es after : " +str(len(V_alphas[i])-1) +" trials.")
```

```
For alpha = 0.2 , it converges after : 20 trials.
For alpha = 0.3 , it converges after : 17 trials.
For alpha = 0.4 , it converges after : 16 trials.
For alpha = 0.5 , it converges after : 15 trials.
For alpha = 0.6 , it converges after : 14 trials.
For alpha = 0.7 , it converges after : 14 trials.
For alpha = 0.8 , it converges after : 14 trials.
For alpha = 0.9 , it converges after : 13 trials.
For alpha = 1.0 , it converges after : 13 trials.
```

alpha=0.9 and 1.0 converge the quickest for E(V)=3

As expected, we can use higher learning rate as the stochasticity of the environment is reduced (reward is always 3 after trial 12) . This makes sense because high learning rate means high confidence in our update and since the stochasticity of the environment is low we are confident with our choices.

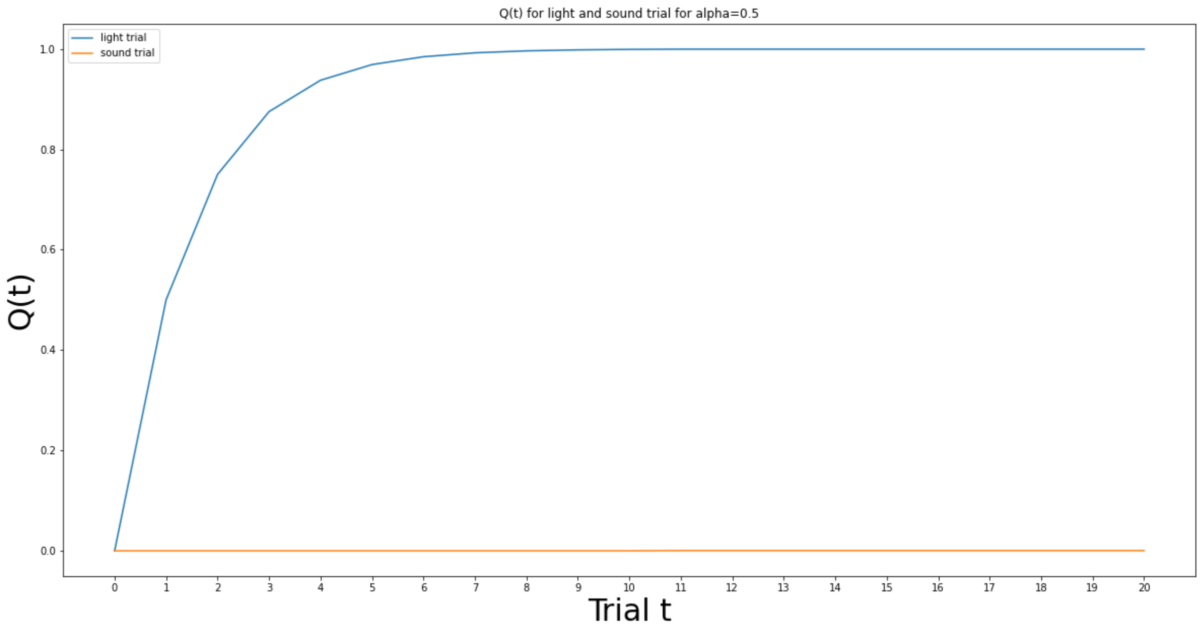
There is an important tradeoff to take into consideration here . High alpha means that we learn very fast but it also means that we will be very sensitive to noise . On the other hand small alpha means that we learn slowly but at least we are less sensitive to noise. That is the core concept in reinforcement learning : tradeoff between stochasticity of environment and speed of learning . In more advanced RL problems, we have the exploration/exploitation dilemma. The stochasticity in RL condition the value of alpha that we should use

In [ ]:

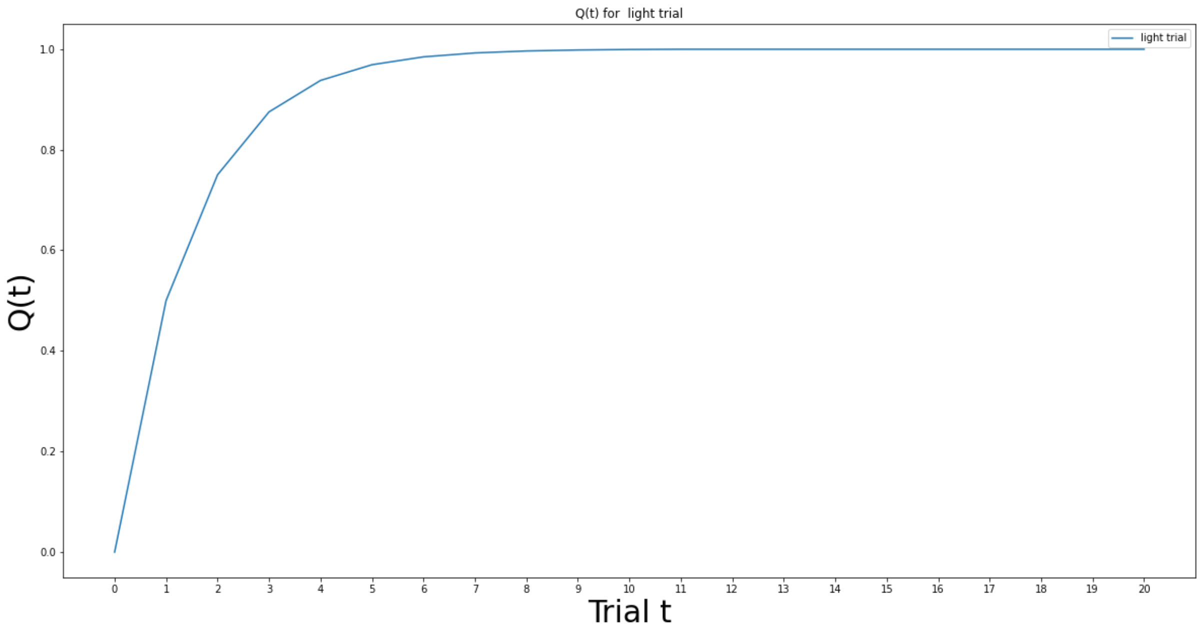
Part 2: Q-Learning

In [67]:
```
alpha=0.5
R = [0]+[1]*20
Q_s_l = [0]
Q_s_s = [0]
Q_both = [0]

#Q=[0]
s_l = [0]+[1]*20
s_s = [0]+[0]*10+[1]*10

for  t in range(1,len(R)):

    if(s_l[t]==1 and s_s[t]==1):

        #Q_both.append(Q_both[t-1] + alpha * (R[t]-(Q_s_l[t-1]+Q_s_s[t-
1])))
        #Q_s_l.append(Q_s_l[t-1])
        #Q_s_s.append(Q_s_s[t-1])
        Q_s_l.append(Q_s_l[t-1] + alpha * (R[t]-(Q_s_l[t-1]+Q_s_s[t-1
])))
        Q_s_s.append(Q_s_s[t-1] + alpha * (R[t]-(Q_s_l[t-1]+Q_s_s[t-1
])))



    elif(s_l[t]==1) :
        #Q_both.append(Q_both[t-1])
        #Q.append(Q[t-1] + alpha * (R[t]-Q_s_l[t-1]))
        Q_s_l.append(Q_s_l[t-1] + alpha * (R[t]-Q_s_l[t-1]))
        Q_s_s.append(Q_s_s[t-1])

    else :
        #Q_both.append(Q_both[t-1])
        #Q.append(Q[t-1] + alpha * (R[t]-Q_s_s[t-1]))
        Q_s_l.append(Q_s_l[t-1])
        Q_s_s.append(Q_s_s[t-1] + alpha * (R[t]-Q_s_s[t-1]))
```

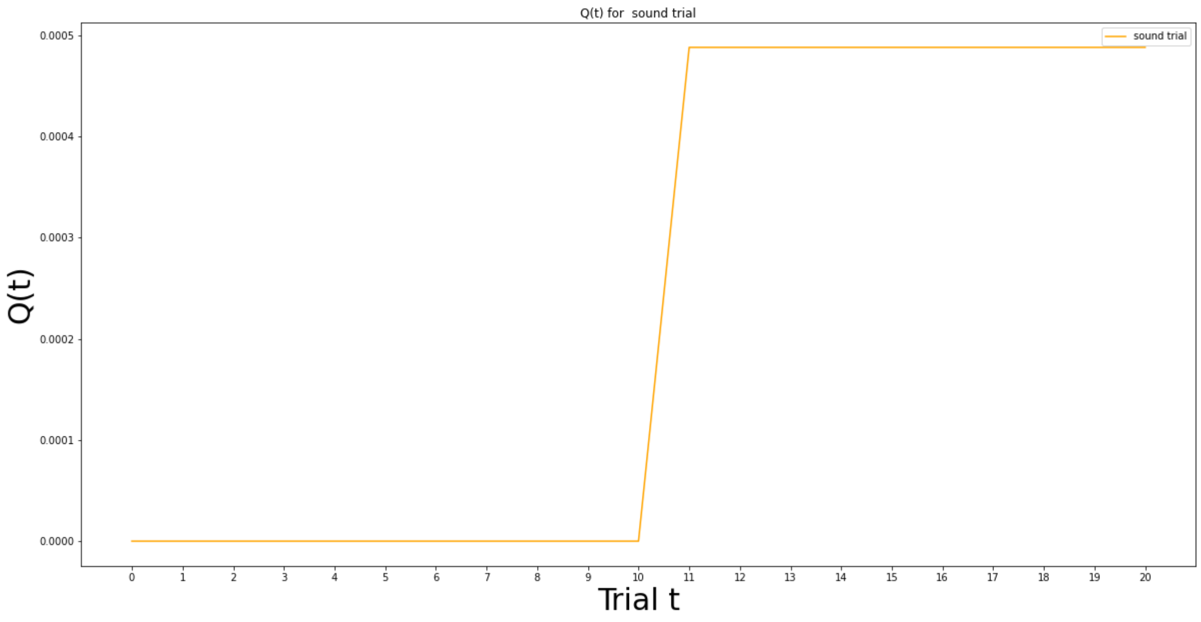
In [59]:
```
plt.plot(Q_s_l,label="light trial")
plt.plot(Q_s_s,label="sound trial")

plt.xlabel("Trial t",fontsize=30)
plt.ylabel("Q(t)",fontsize=30)
plt.xticks(range(0,21))
plt.title ("Q(t) for light and sound trial for alpha=0.5")
plt.legend()
plt.show()



# Plot them one by one so  that we see the effect of sound trial (very l
ow with alpha=0.5)
plt.plot(Q_s_l,label="light trial")
plt.xlabel("Trial t",fontsize=30)
plt.ylabel("Q(t)",fontsize=30)
plt.xticks(range(0,21))
plt.title ("Q(t) for  light trial")
plt.legend()
plt.show()

plt.plot(Q_s_s,label="sound trial",color='orange')
plt.xlabel("Trial t",fontsize=30)
plt.ylabel("Q(t)",fontsize=30)
plt.xticks(range(0,21))
plt.title ("Q(t) for  sound trial")
plt.legend()
plt.show()
```



Q(t) for light and sound trial for alpha=0.5
light trial
sound trial
Q(t)
Trial t
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20



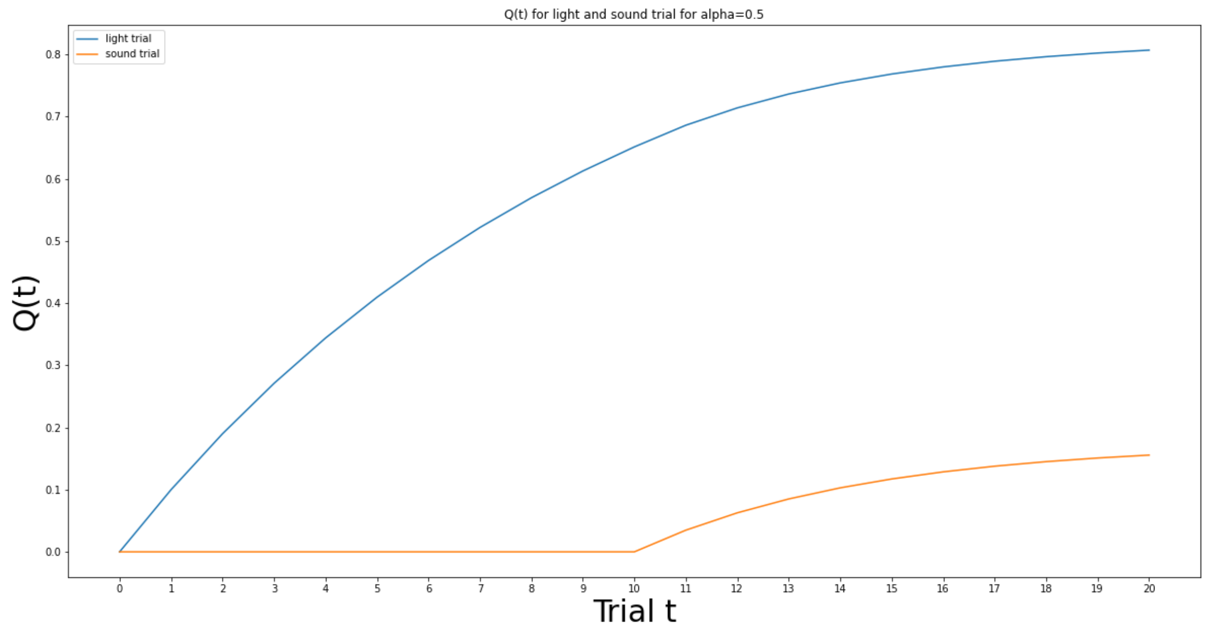
Q(t) for light trial
light trial
Q(t)
Trial t



Q(t) for sound trial
sound trial
Q(t)
Trial t
0.0005
0.0004
0.0003
0.0002
0.0001
0.0000

Let's decrease alpha (learn more slowly so that the effect of sound trial is bigger since we will still be learning)

In [62]:
```
alpha=0.1
R = [0]+[1]*20
Q_s_l = [0]
Q_s_s = [0]
Q_both = [0]

#Q=[0]
s_l = [0]+[1]*20
s_s = [0]+[0]*10+[1]*10
for  t in range(1,len(R)):

    if(s_l[t]==1 and s_s[t]==1):

        #Q_both.append(Q_both[t-1] + alpha * (R[t]-(Q_s_l[t-1]+Q_s_s[t-
1])))
        #Q_s_l.append(Q_s_l[t-1])
        #Q_s_s.append(Q_s_s[t-1])
        Q_s_l.append(Q_s_l[t-1] + alpha * (R[t]-(Q_s_l[t-1]+Q_s_s[t-1
])))
        Q_s_s.append(Q_s_s[t-1] + alpha * (R[t]-(Q_s_l[t-1]+Q_s_s[t-1
])))



    elif(s_l[t]==1) :
        #Q_both.append(Q_both[t-1])
        #Q.append(Q[t-1] + alpha * (R[t]-Q_s_l[t-1]))
        Q_s_l.append(Q_s_l[t-1] + alpha * (R[t]-Q_s_l[t-1]))
        Q_s_s.append(Q_s_s[t-1])
    else :
        #Q_both.append(Q_both[t-1])
        #Q.append(Q[t-1] + alpha * (R[t]-Q_s_s[t-1]))
        Q_s_l.append(Q_s_l[t-1])
        Q_s_s.append(Q_s_s[t-1] + alpha * (R[t]-Q_s_s[t-1]))

plt.plot(Q_s_l,label="light trial")
plt.plot(Q_s_s,label="sound trial")

plt.xlabel("Trial t",fontsize=30)
plt.ylabel("Q(t)",fontsize=30)
plt.xticks(range(0,21))
plt.title ("Q(t) for light and sound trial for lower alpha=0.1")
plt.legend()
plt.show()
```

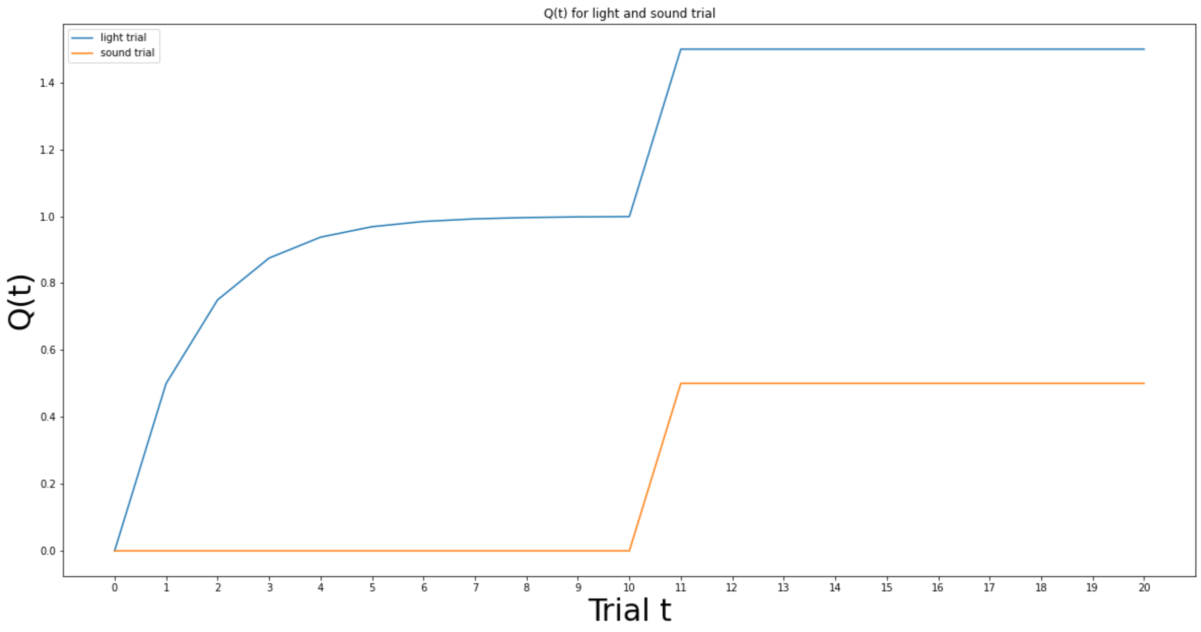


As expected , decreasing alpha , increase the effect for sound trial because we learn more slowly and therefore learning is not done when the sound trial starts to be activated.

Delta is a reward prediction error that account for sound trial or light trial or both. Delta can be very small and it seems like there is no update . This is why I contrasted with a small value of alpha (0.1) because it will have a different effect on the delta values.

In [73]:
```
alpha=0.5
R = [0]+[1]*10+[2]*10
Q_s_l = [0]
Q_s_s = [0]
Q_both = [0]

#Q=[0]
s_l = [0]+[1]*20
s_s = [0]+[0]*10+[1]*10
for  t in range(1,len(R)):
    if(s_l[t]==1 and s_s[t]==1):

        #Q_both.append(Q_both[t-1] + alpha * (R[t]-(Q_s_l[t-1]+Q_s_s[t-
1])))
        #Q_s_l.append(Q_s_l[t-1])
        #Q_s_s.append(Q_s_s[t-1])
        Q_s_l.append(Q_s_l[t-1] + alpha * (R[t]-(Q_s_l[t-1]+Q_s_s[t-1
])))
        Q_s_s.append(Q_s_s[t-1] + alpha * (R[t]-(Q_s_l[t-1]+Q_s_s[t-1
])))
    elif(s_l[t]==1) :
        Q_both.append(Q_both[t-1])
        #Q.append(Q[t-1] + alpha * (R[t]-Q_s_l[t-1]))
        Q_s_l.append(Q_s_l[t-1] + alpha * (R[t]-Q_s_l[t-1]))
        Q_s_s.append(Q_s_s[t-1])
    else :
        #Q_both.append(Q_both[t-1])
        #Q.append(Q[t-1] + alpha * (R[t]-Q_s_s[t-1]))
        Q_s_l.append(Q_s_l[t-1])
        Q_s_s.append(Q_s_s[t-1] + alpha * (R[t]-Q_s_s[t-1]))

plt.plot(Q_s_l,label="light trial")
plt.plot(Q_s_s,label="sound trial")
#plt.plot(Q_both,label="light and sound trial")

plt.xlabel("Trial t",fontsize=30)
plt.ylabel("Q(t)",fontsize=30)
plt.xticks(range(0,21))
plt.title ("Q(t) for light and sound trial")
plt.legend()
```

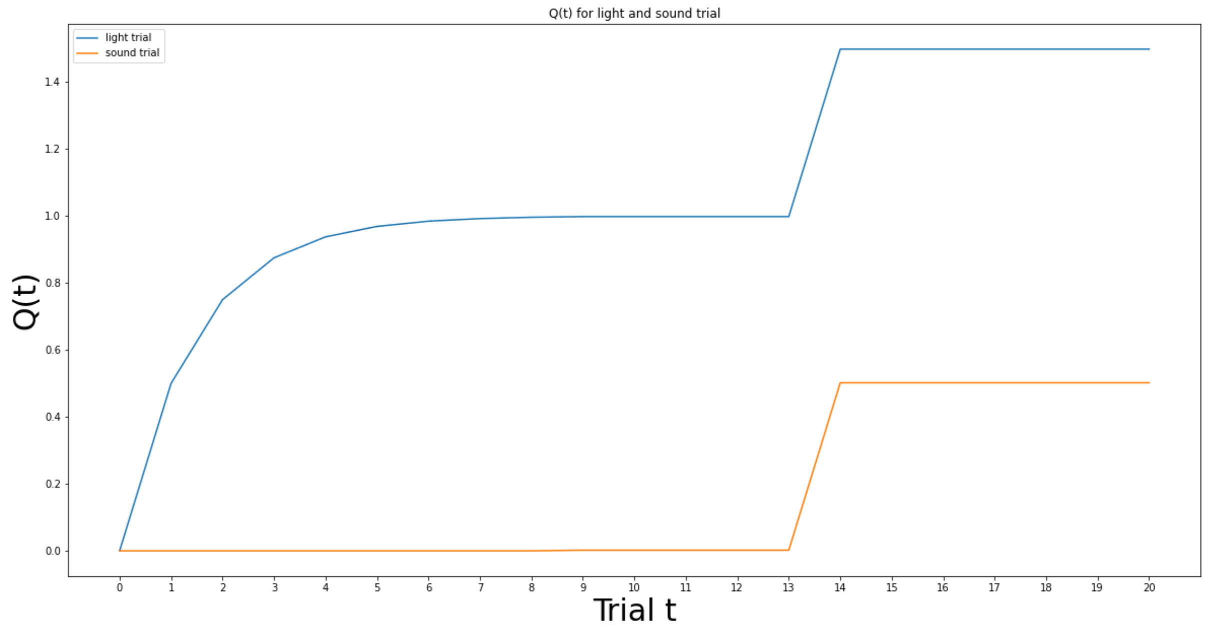
```
Out[73]: <matplotlib.legend.Legend at 0x1196d9490>
```



The stochasticity of the environment has been increased in this environment.

In [75]:
```
alpha=0.5
R = [0]+[1]*13+[2]*7
Q_s_l = [0]
Q_s_s = [0]
Q_both = [0]

#Q=[0]
s_l = [0]+[1]*20
s_s = [0]+[0]*8+[1]*12
for  t in range(1,len(R)):
    if(s_l[t]==1 and s_s[t]==1):

        #Q_both.append(Q_both[t-1] + alpha * (R[t]-(Q_s_l[t-1]+Q_s_s[t-
1])))
        #Q_s_l.append(Q_s_l[t-1])
        #Q_s_s.append(Q_s_s[t-1])
        Q_s_l.append(Q_s_l[t-1] + alpha * (R[t]-(Q_s_l[t-1]+Q_s_s[t-1
])))
        Q_s_s.append(Q_s_s[t-1] + alpha * (R[t]-(Q_s_l[t-1]+Q_s_s[t-1
])))
    elif(s_l[t]==1) :
        #Q_both.append(Q_both[t-1])
        #Q.append(Q[t-1] + alpha * (R[t]-Q_s_l[t-1]))
        Q_s_l.append(Q_s_l[t-1] + alpha * (R[t]-Q_s_l[t-1]))
        Q_s_s.append(Q_s_s[t-1])
    else :
        #Q_both.append(Q_both[t-1])
        #Q.append(Q[t-1] + alpha * (R[t]-Q_s_s[t-1]))
        Q_s_l.append(Q_s_l[t-1])
        Q_s_s.append(Q_s_s[t-1] + alpha * (R[t]-Q_s_s[t-1]))

plt.plot(Q_s_l,label="light trial")
plt.plot(Q_s_s,label="sound trial")
#plt.plot(Q_both,label="light and sound trial")

plt.xlabel("Trial t",fontsize=30)
plt.ylabel("Q(t)",fontsize=30)
plt.xticks(range(0,21))
plt.title ("Q(t) for light and sound trial")
plt.legend()
```

```
Out[75]: <matplotlib.legend.Legend at 0x1197b9690>
```



The stochasticity of the environment has been increased compare to original environment but is lower than the previous environment.

The trial experiment has changed explaining the shift for Qss and Qsl due to the fact that sound trial activates only after 13 trials compare to 10 in the previous environments.

Overall, it seems that the time delay between two different type of trials influences a lot the learning of the 2 types of trials.

```
In [ ]:
```