```python
import numpy as np

def J(w):
    return w[0]**2 + w[1]**2 + 4*w[0] - 6*w[1] - 7

def gradientJ(w):
    return np.array([2.0*w[0]+4, 2.0*w[1]-6])

# Do the iteration specified in the iterate-parameter
# Return the updated w
# Note: If you want to get J(w) of fifth itertion, use the updated w from fourth
def wUpdatedIterate(wCurrent, alpha, iterate):

  print("%-2s %-18s %-18s %-11s %-18s"
      % ("k", "w", "gradient J", "J(w)", "new w"))
  print("------------------------------------------------------------")

  w = wCurrent
  for k in range(1, iterate+1):
    print("%02d %-18s %-18s %07.4f "
          % (k, np.round(w,4), np.round(gradientJ(w),4), J(w)), end="")
    w = w - alpha * gradientJ(w)
    print(" %-18s" % (np.round(w,4)))

  return w
```

Questions 1 to 6

- Go to https://www.dcode.fr/minimum-function
- Add **(x)^2+(y)^2+4*x-6*y-7** as "FUNCTION (F(X)=)"
- Add **x,y** as 'WITH RESPECT TO'
- click *CALCULATE MINIMUM* button

```python
w50th = wUpdatedIterate(np.array([5, 5]), 0.3, 49) # [5,5] is the value after fir
```

```
print("ANSWER#1 min(J(w)): ", J(w50th))
print("ANSWER#2 w1: ", w50th[0])
print("ANSWER#3 w2: ", w50th[1])
```

```
w5th = wUpdatedIterate(np.array([5, 5]), 0.3, 4) # first iteration done, do rest
print("------------")
print("ANSWER#4 w1: ", w5th[0])
print("ANSWER#5 w2: ", w5th[1])
```

```
w4th = wUpdatedIterate(np.array([5, 5]), 0.3, 3)
w5th = wUpdatedIterate(np.array([5, 5]), 0.3, 4)
print("------------")
print("ANSWER#6 J5th(w)-J4th(w) :", round(abs(J(w5th)-J(w4th)), 4))
```

- The accepted answer is **0.1823**
- The above code gets **0.18235** and rounded up to **0.1824**.
- Most common method of rounding rule is: round up if the next digit is less than 5. Round up otherwise.

Questions 7 to 10

```
import numpy as np
from numpy import linalg as LA
x = np.array([
    [1, 0],
    [1, 0.25],
    [1, 0.5],
    [1, 0.75],
    [1, 1.00]
]) # adding 1 in every inputs in order to accomodate bias/intercept.

y = np.array([
    [0.8822],
    [1.2165],
    [1.3171],
    [1.7930],
    [1.9826]
])


def J2(w):
    return 1.0/10*sum([(y[i][0]-w[1]*x[i][1]-w[0])**2 for i in range(5)])

def gradientJ2(w):
    return np.array([1.0/10*sum([2*(y[i][0]-w[1]*x[i][1]-w[0])*-1 for i in range(
                     1.0/10*sum([2*(y[i][0]-w[1]*x[i][1]-w[0])*-x[i][1] for i in
                    ])

# Do the iteration specified in the iterate-parameter
# Return the updated w
# Note: If you want to get J(w) of fifth itertion, use the updated w from fourth
def wUpdatedIterate2(wCurrent, alpha, iterate):

  print("%-2s %-18s %-18s %-12s %-18s"
      % ("k", "w", "gradient J2", "J2(w)", "new w"))
  print("------------------------------------------------------------")

  w = wCurrent
  for k in range(1, iterate+1):
    print("%02d %-18s %-18s %8.4f "
          % (k, np.round(w,4), np.round(gradientJ2(w),4), J2(w)), end="")
    w = w - alpha * gradientJ2(w)
    print(" %-16s" % (np.round(w,4)))

  return w
```

```
w1000th = wUpdatedIterate2(np.array([0, 0]), .3, 999) # first iteration done, do
```

```
# normal equation
wAccordingToNormalEquation = LA.inv(x.T.dot(x)).dot(x.T).dot(y)

print("W according to normal equation: %s " % wAccordingToNormalEquation)

from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(x[:,1:],y) # Remove all 1's added for accomodating bias/intercept
wAccordingToLinearRegression = lr.intercept_,lr.coef_
print("W according to linear equation: %s %s" % wAccordingToLinearRegression)
print("W according to gradient descnt: %s" % w1000th)
```

```
w5th = wUpdatedIterate2(np.array([0, 0]), 1, 4)
print("------------")
print("ANSWER#7 :", len(w5th))
print("ANSWER#8 w0 :", round(w5th[0], 4))
print("ANSWER#9 w1 :", round(w5th[1], 4))
```

```python
w5th = wUpdatedIterate2(np.array([0, 0]), 1, 4)
print("ANSWER10 J5th(w) :", round(J2(w5th), 4))
```

```python
w6th = wUpdatedIterate2(np.array([0, 0]), 1, 6)
```