warmups.py

```python
import numpy as np


def w1(X):
    """
    Input:
    - X: A numpy array

    Returns:
    - A matrix Y such that Y[i, j] = X[i, j] * 10 + 100

    Hint: Trust that numpy will do the right thing
    """
    return X * 10 + 100


def w2(X, Y):
    """
    Inputs:
    - X: A numpy array of shape (N, N)
    - Y: A numpy array of shape (N, N)

    Returns:
    A numpy array Z such that Z[i, j] = X[i, j] + 10 * Y[i, j]

    Hint: Trust that numpy will do the right thing
    """
    return X + 10 * Y


def w3(X, Y):
    """
    Inputs:
    - X: A numpy array of shape (N, N)
    - Y: A numpy array of shape (N, N)
```

```
    Returns:
    A numpy array Z such that Z[i, j] = X[i, j] * Y[i, j] - 10

    Hint: By analogy to +, * will do the same thing
    """
    return X * Y - 10


def w4(X, Y):
    """
    Inputs:
    - X: Numpy array of shape (N, N)
    - Y: Numpy array of shape (N, N)

    Returns:
    A numpy array giving the matrix product X times Y

    Hint: Not the same as *!
    """
    return X.dot(Y)


def w5(X):
    """
    Inputs:
    - X: A numpy array of shape (N, N) of floating point numbers

    Returns:
    A numpy array with the same data as M, but cast to 32-bit integers

    Hint: astype
    """
    return X.astype(np.int32)


def w6(X, Y):
    """
    Inputs:
    - X: A numpy array of shape (N,) of integers
```

```
    - Y: A numpy array of shape (N,) of integers

    Returns:
    A numpy array Z such that Z[i] = float(X[i]) / float(Y[i])

    Hint: Dividing two integers is not the same as dividing two floats
    """
    return X.astype(float) / Y.astype(float)


def w7(X):
    """
    Inputs:
    - X: A numpy array of shape (N, M)

    Returns:
    - A numpy array Y of shape (N * M, 1) containing the entries of X in row
      order. That is, X[i, j] = Y[i * M + j, 0]

    Hint:
    1) np.reshape
    2) You can specify an unknown dimension as -1
    """
    return np.reshape(X, (X.shape[0] * X.shape[1], 1))


def w8(N):
    """
    Inputs:
    - N: An integer

    Returns:
    A numpy array of shape (N, 2N)

    Hint: The error "data type not understood" means you probably called
    np.ones or np.zeros with two arguments, instead of a tuple for the shape
    """
    return np.zeros((N, 2 * N))
```

```python
def w9(X):
    """
    Inputs:
    - X: A numpy array of shape (N, M) where each entry is between 0 and 1

    Returns:
    A numpy array Y where Y[i, j] = True if X[i, j] > 0.5

    Hint: Trust python to do the right thing
    """
    return X > 0.5


def w10(N):
    """
    Inputs:
    - N: An integer

    Returns:
    A numpy array X of shape (N,) such that X[i] = i

    Hint: np.arange
    """
    return np.arange(N)


def w11(A, v):
    """
    Inputs:
    - A: A numpy array of shape (N, F)
    - v: A numpy array of shape (F, 1)

    Returns:
    Numpy array of shape (N, 1) giving the matrix-vector product Av
    """
    return A.dot(v)
```

```python
def w12(A, v):
    """
    Inputs:
    - A: A numpy array of shape (N, N), of full rank
    - v: A numpy array of shape (N, 1)

    Returns:
    Numpy array of shape (N, 1) giving the matrix-vector product of the inverse
    of A and v: A^-1 v
    """
    return np.linalg.inv(A).dot(v)


def w13(u, v):
    """
    Inputs:
    - u: A numpy array of shape (N, 1)
    - v: A numpy array of shape (N, 1)

    Returns:
    The inner product u^T v

    Hint: .T
    """
    return u.T.dot(v)


def w14(v):
    """
    Inputs:
    - v: A numpy array of shape (N, 1)

    Returns:
    The L2 norm of v: norm = (sum_i^N v[i]^2)^(1/2)
    You MAY NOT use np.linalg.norm
    """
    return np.sqrt(np.sum(v ** 2))
```

```python
def w15(X, i):
    """
    Inputs:
    - X: A numpy array of shape (N, M)
    - i: An integer in the range 0 <= i < N

    Returns:
    Numpy array of shape (M,) giving the ith row of X
    """
    return X[i]


def w16(X):
    """
    Inputs:
    - X: A numpy array of shape (N, M)

    Returns:
    The sum of all entries in X

    Hint: np.sum
    """
    return np.sum(X)


def w17(X):
    """
    Inputs:
    - X: A numpy array of shape (N, M)

    Returns:
    A numpy array S of shape (N,) where S[i] is the sum of row i of X

    Hint: np.sum has an optional "axis" argument
    """
    return np.sum(X, axis=1)


def w18(X):
```

```
    """
    Inputs:
    - X: A numpy array of shape (N, M)

    Returns:
    A numpy array S of shape (M,) where S[j] is the sum of column j of X

    Hint: Same as above
    """
    return np.sum(X, axis=0)


def w19(X):
    """
    Inputs:
    - X: A numpy array of shape (N, M)

    Returns:
    A numpy array S of shape (N, 1) where S[i, 0] is the sum of row i of X

    Hint: np.sum has an optional "keepdims" argument
    """
    return np.sum(X, axis=1, keepdims=True)


def w20(X):
    """
    Inputs:
    - X: A numpy array of shape (N, M)

    Returns:
    A numpy array S of shape (N, 1) where S[i] is the L2 norm of row i of X
    """
    return np.array([np.linalg.norm(X, axis=1)]).T
```

tests.py

```
import numpy as np



def t1(L):
    """
    Inputs:
    - L: A list of M numpy arrays, each of shape (1, N)

    Returns:
    A numpy array of shape (M, N) giving all inputs stacked together

    Par: 1 line
    Instructor: 1 line

    Hint: vstack/hstack/dstack, no for loop
    """
    return np.vstack(L)



def t2(X):
    """
    Inputs:
    - X: A numpy array of shape (N, N)

    Returns:
    Numpy array of shape (N,) giving the eigenvector corresponding to the
    smallest eigenvalue of X

    Par: 5 lines
    Instructor: 3 lines

    Hints:
    1) np.linalg.eig
    2) np.argmin
    3) Watch rows and columns!
    """
```

```
    merp, e_val = np.linalg.eig(X)
    min_val = np.argmin(merp)
    return e_val[:,min_val]


def t3(X):
    """
    Inputs:
    - A: A numpy array of any shape

    Returns:
    A copy of X, but with all negative entires set to 0

    Par: 3 lines
    Instructor: 2 lines

    Hint:
    1) If S is a boolean array with the same shape as X, then X[S] gives an
       array containing all elements of X corresponding to true values of S
    2) X[S] = v assigns the value v to all entires of X corresponding to
       true values of S.
    """
    return np.where(X < 0, 0, X)


def t4(R, X):
    """
    Inputs:
    - R: A numpy array of shape (3, 3) giving a rotation matrix
    - X: A numpy array of shape (N, 3) giving a set of 3-dimensional vectors

    Returns:
    A numpy array Y of shape (N, 3) where Y[i] is X[i] rotated by R

    Par: 3 lines
    Instructor: 1 line

    Hint:
    1) If v is a vector, then the matrix-vector product Rv rotates the vector
```

```
      by the matrix R.
   2) .T gives the transpose of a matrix
   """
   return X.dot(R.T)


def t5(X):
   """
   Inputs:
   - X: A numpy array of shape (N, N)

   Returns:
   A numpy array of shape (4, 4) giving the upper left 4x4 submatrix of X
   minus the bottom right 4x4 submatrix of X.

   Par: 2 lines
   Instructor: 1 line

   Hint:
   1) X[y0:y1, x0:x1] gives the submatrix
      from rows y0 to (but not including!) y1
      from columns x0 (but not including!) x1
   """
   return X[0:4, 0:4] - X[-4:, -4:]


def t6(N):
   """
   Inputs:
   - N: An integer

   Returns:
   A numpy array of shape (N, N) giving all 1s, except the first and last 5
   rows and columns are 0.

   Par: 6 lines
   Instructor: 3 lines
   """
   X = np.ones((N, N))
```

```
    X[:, -5:] = 0
    X[:, 0:5] = 0
    X[-5:] = 0
    X[0:5,] = 0
    return X


def t7(X):
    """
    Inputs:
    - X: A numpy array of shape (N, M)

    Returns:
    A numpy array Y of the same shape as X, where Y[i] is a vector that points
    the same direction as X[i] but has unit norm.

    Par: 3 lines
    Instructor: 1 line

    Hints:
    1) The vector v / ||v||| is the unit vector pointing in the same direction
       as v (as long as v != 0)
    2) Divide X by a vector of normalization factors, of shape (N, 1)
    3) Elementwise operations between an array of shape (N, M) and an array of
       shape (N, 1) work -- try it! This is called "broadcasting"
    4) Elementwise operations between an array of shape (N, M) and an array of
       shape (N,) won't work -- try reshaping
    """
    reshaped = np.reshape(np.linalg.norm(X, axis=1), (-1, 1))
    return X / reshaped


def t8(X):
    """
    Inputs:
    - X: A numpy array of shape (N, M)

    Returns:
    A numpy array Y of shape (N, M) where Y[i] contains the same data as X[i],
```

```
    but normalized to have mean 0 and standard deviation 1.

    Par: 3 lines
    Instructor: 1 line
    """
    num = X - (X.mean(axis = 1, keepdims=True))
    den = X.std(axis=1, keepdims=True)
    return np.divide(num, den)


def t9(q, k, v):
    """
    Inputs:
    - q: A numpy array of shape (1, K) (queries)
    - k: A numpy array of shape (N, K) (keys)
    - v: A numpy array of shape (N, 1) (values)

    Returns:
    sum_i exp(-||q-k_i||^2) * v[i]

    Par: 3 lines
    Instructor: 1 ugly line

    Hints:
    1) You can perform elementwise operations on arrays of shape (N, K) and
       (1, K) with broadcasting
    2) Recall that np.sum has useful "axis" and "keepdims" options
    3) np.exp and friends apply elementwise to arrays
    """
    inner = np.exp(-1 * np.linalg.norm(q - k, axis=1) ** 2)
    return np.sum(inner.dot(v), axis=0)


def t10(Xs):
    """
    Inputs:
    - Xs: A list of length L, containing numpy arrays of shape (N, M)

    Returns:
```

```
    A numpy array R of shape (L, L) where R[i, j] is the Euclidean distance
    between C[i] and C[j], where C[i] is an M-dimensional vector giving the
    centroid of Xs[i]

    Par: 12 lines
    Instructor: 3 lines (after some work!)

    Hints:
    1) You can use a for loop over L
    2) Distances are symmetric
    3) Go one step at a time
    4) Our 3-line solution uses no loops, and uses the algebraic trick from the
       next problem.
    """

    N = len(Xs)
    c = np.mean(Xs, axis=1)
    res = np.zeros((N, N))

    c_height = len(c)
    for x in range(0, c_height):
        merp = []
        for y in range(0, c_height):
            diff = c[x] - c[y]
            distance = np.linalg.norm(diff)
            merp.append(distance)
        res[x] = np.asarray(merp)

    return res


def t11(X):
    """
    Inputs:
    - X: A numpy array of shape (N, M)

    Returns:
    A numpy array D of shape (N, N) where D[i, j] gives the Euclidean distance
    between X[i] and X[j], using the identity
```

```
    ||x - y||^2 = ||x||^2 + ||y||^2 - 2x^T y

    Par: 3 lines
    Instructor: 2 lines (you can do it in one but it's wasteful compute-wise)

    Hints:
    1) What happens when you add two arrays of shape (1, N) and (N, 1)?
    2) Think about the definition of matrix multiplication
    3) Transpose is your friend
    4) Note the square! Use a square root at the end
    5) On some machines, ||x||^2 + ||x||^2 - 2x^Tx may be slightly negative,
       causing the square root to crash. Just take max(0, value) before the
       square root. Seems to occur on Macs.
    """

    res = (np.linalg.norm(X, axis=1, keepdims=True).T ** 2) + (np.linalg.norm(X,
axis=1, keepdims=True) ** 2) - (2 * (X).dot(X.T))
    return np.sqrt(np.where(res < 0, 0, res))


def t12(X, Y):
    """
    Inputs:
    - X: A numpy array of shape (N, F)
    - Y: A numpy array of shape (M, F)

    Returns:
    A numpy array D of shape (N, M) where D[i, j] is the Euclidean distance
    between X[i] and Y[i].

    Par: 3 lines
    Instructor: 2 lines (you can do it in one, but it's more than 80 characters
                with good code formatting)

    Hints: Similar to previous problem
    """
    res = ((np.linalg.norm(X, axis=1, keepdims=True).T ** 2) + (np.linalg.norm(Y,
axis=1, keepdims=True) ** 2)).T - (2 * X.dot(Y.T))
    return np.sqrt(np.where(res < 0, 0, res))
```

```
def t13(q, V):
    """
    Inputs:
    - q: A numpy array of shape (1, M) (query)
    - V: A numpy array of shape (N, M) (values)

    Return:
    The index i that maximizes the dot product q . V[i]

    Par: 1 line
    Instructor: 1 line

    Hint: np.argmax
    """
    return np.argmax(q.dot(V.T))


def t14(X, y):
    """
    Inputs:
    - X: A numpy array of shape (N, M)
    - y: A numpy array of shape (N, 1)

    Returns:
    A numpy array w of shape (M, 1) such that ||y - Xw||^2 is minimized

    Par: 2 lines
    Instructor: 1 line

    Hint: np.linalg.lstsq, or use the pseudoinverse (X^T X)^-1 X^T y
    """
    res = np.linalg.lstsq(X, y, None)
    return res[0]


def t15(X, Y):
    """
```

```
    Inputs:
    - X: A numpy array of shape (N, 3)
    - Y: A numpy array of shape (N, 3)

    Returns:
    A numpy array C of shape (N, 3) such C[i] is the cross product between X[i]
    and Y[i]

    Par: 1 line
    Instructor: 1 line

    Hint: np.cross
    """
    return np.cross(X, Y)


def t16(X):
    """
    Inputs:
    - X: A numpy array of shape (N, M)

    Returns:
    A numpy array Y of shape (N, M - 1) such that
    Y[i, j] = X[i, j] / X[i, M - 1]
    for all 0 <= i < N and all 0 <= j < M - 1

    Par: 1 line
    Instructur: 1 line

    Hints:
    1) If it doesn't broadcast, reshape or np.expand_dims
    2) X[:, -1] gives the last column of X
    """
    # could be done in one line
    dummy_X = X[:, -1].reshape(((np.ma.size(X, axis=0)), 1))
    div_res = np.divide(X, dummy_X)
    return np.delete(div_res, -1, axis=1)
```

```
def t17(X):
    """
    Inputs:
    - X: A numpy array of shape (N, M)

    Returns:
    A numpy array Y of shape (N, M + 1) such that
        Y[i, :F] = X[i]
        Y[i, F] = 1

    Par: 1 line
    Instructor: 1 line

    Hint: np.hstack, np.ones
    """
    res = np.ones((np.ma.size(X,axis=0),1))
    return np.hstack((X, res))


def t18(N, r, x, y):
    """
    Inputs:
    - N: An integer
    - r: A floating-point number
    - x: A floating-point number
    - y: A floating-point number

    Returns:
    A numpy array I of floating point numbers and shape (N, N) such that:
    I[i, j] = 1 if ||(j, i) - (x, y)|| < r
    I[i, j] = 0 otherwise

    Par: 3 lines
    Instructor: 2 lines

    Hints:
    1) np.meshgrid and np.arange give you X, Y
    2) Arrays have an astype method
    """
```

```
    return None


def t19(N, s, x, y):
    """
    Inputs:
    - N: An integer
    - s: A floating-point number
    - x: A floating-point number
    - y: A floating-point number

    Returns:
    A numpy array I of shape (N, N) such that
    I[i, j] = exp(-||(j, i) - (x, y)||^2 / s^2)

    Par: 3 lines
    Instructor: 2 lines

    Hint: Be careful with types -- float and int aren't the same!
    """
    return None


def t20(N, v):
    """
    Inputs:
    - N: An integer
    - v: A numpy array of shape (3,) giving coefficients v = [a, b, c]

    Returns:
    A numpy array of shape (N, N) such that M[i, j] is the distance between the
    point (j, i) and the line a*j + b*i + c = 0

    Par: 4 lines
    Instructor: 2 lines

    Hints:
    1) The distance between the point (x, y) and the line ax+by+c=0 is given by
       abs(ax + by + c) / sqrt(a^2 + b^2)
```

```
    (The sign of the numerator tells which side the point is on)
2) np.abs
"""
return None
```

python run.py --alltests

Running t1
Running t2
Running t3
Running t4
Running t5
Running t6
Running t7
Running t8
Running t9
Running t10
Running t11
Running t12
Running t13
Running t14
Running t15
Running t16
Running t17
Running t18
Not implemented t18 or returned None on seed 442
Running t19
Not implemented t19 or returned None on seed 442
Running t20
Not implemented t20 or returned None on seed 442
Ran all tests
17/20 = 85.0

python run.py --allwarmups

Running w1
Running w2
Running w3
Running w4
Running w5
Running w6
Running w7
Running w8
Running w9
Running w10
Running w11
Running w12
Running w13
Running w14
Running w15
Running w16
Running w17
Running w18
Running w19
Running w20
Ran warmup tests
20/20 = 100.0