```
import numpy as np

print("TIL Python 3.5 added '@' as an infix operator for the dot product of numpy / sympy arrays.")

def printer(*args):
    for arg in args:
        print(arg, eval(arg, locals(), globals()))

n = 5
m = 7

v = np.random.randint(low=1,high=5,size=(n))
w = np.random.randint(low=1,high=5,size=(n))
printer('v','w')
printer('v@w','v.T@w', 'w.T@v')

A = np.random.randint(low=1,high=5,size=(n,m))
B = np.random.randint(low=1,high=5,size=(m,n))

printer('v.T@(A@B)@w', 'w.T@(A@B).T@v')
printer('(A@B).T', 'B.T@A.T')

ANSWER = """
TIL Python 3.5 added '@' as an infix operator for the dot product of numpy / sympy arrays.
v [3 3 1 3 4]
w [2 1 1 2 2]
v@w 24
v.T@w 24
w.T@v 24
v.T@(A@B)@w 4523
w.T@(A@B).T@v 4523
(A@B).T [[61 53 64 39 33]
 [52 53 62 37 32]
 [42 40 48 25 23]
 [54 57 63 35 33]
 [41 33 42 29 21]]
B.T@A.T [[61 53 64 39 33]
 [52 53 62 37 32]
 [42 40 48 25 23]
 [54 57 63 35 33]
 [41 33 42 29 21]]
"""
```

```
import numpy as np

print("TIL Python 3.5 added '@' as an infix operator for the dot product of numpy / sympy arrays.")

def printer(*args):
    for arg in args:
        print(arg, eval(arg, locals(), globals()))

n = 5
m = 7

v = np.random.randint(low=1,high=5,size=(n))
w = np.random.randint(low=1,high=5,size=(n))
printer('v','w')
printer('v@w','v.T@w', 'w.T@v')

A = np.random.randint(low=1,high=5,size=(n,m))
B = np.random.randint(low=1,high=5,size=(m,n))

printer('v.T@(A@B)@w', 'w.T@(A@B).T@v')
printer('(A@B).T', 'B.T@A.T')

ANSWER = """
TIL Python 3.5 added '@' as an infix operator for the dot product of numpy / sympy arrays.
v [3 3 1 3 4]
w [2 1 1 2 2]
v@w 24
v.T@w 24
w.T@v 24
v.T@(A@B)@w 4523
w.T@(A@B).T@v 4523
(A@B).T [[61 53 64 39 33]
 [52 53 62 37 32]
 [42 40 48 25 23]
 [54 57 63 35 33]
 [41 33 42 29 21]]
B.T@A.T [[61 53 64 39 33]
 [52 53 62 37 32]
 [42 40 48 25 23]
 [54 57 63 35 33]
 [41 33 42 29 21]]
"""
```

```
import numpy as np

print("TIL Python 3.5 added '@' as an infix operator for the dot product of numpy / sympy arrays.")

def printer(*args):
    for arg in args:
        print(arg, eval(arg, locals(), globals()))

n = 5
m = 7

v = np.random.randint(low=1,high=5,size=(n))
w = np.random.randint(low=1,high=5,size=(n))
printer('v','w')
printer('v@w','v.T@w', 'w.T@v')

A = np.random.randint(low=1,high=5,size=(n,m))
B = np.random.randint(low=1,high=5,size=(m,n))

printer('v.T@(A@B)@w', 'w.T@(A@B).T@v')
printer('(A@B).T', 'B.T@A.T')

ANSWER = """
TIL Python 3.5 added '@' as an infix operator for the dot product of numpy / sympy arrays.
v [3 3 1 3 4]
w [2 1 1 2 2]
v@w 24
v.T@w 24
w.T@v 24
v.T@(A@B)@w 4523
w.T@(A@B).T@v 4523
(A@B).T [[61 53 64 39 33]
 [52 53 62 37 32]
 [42 40 48 25 23]
 [54 57 63 35 33]
 [41 33 42 29 21]]
B.T@A.T [[61 53 64 39 33]
 [52 53 62 37 32]
 [42 40 48 25 23]
 [54 57 63 35 33]
 [41 33 42 29 21]]
"""
```

```python
import numpy as np

print("TIL Python 3.5 added '@' as an infix operator for the dot product of numpy / sympy arrays.")

def printer(*args):
    for arg in args:
        print(arg, eval(arg, locals(), globals()))

n = 5
m = 7

v = np.random.randint(low=1,high=5,size=(n))
w = np.random.randint(low=1,high=5,size=(n))
printer('v','w')
printer('v@w','v.T@w', 'w.T@v')

A = np.random.randint(low=1,high=5,size=(n,m))
B = np.random.randint(low=1,high=5,size=(m,n))

printer('v.T@(A@B)@w', 'w.T@(A@B).T@v')
printer('(A@B).T', 'B.T@A.T')

ANSWER = """
TIL Python 3.5 added '@' as an infix operator for the dot product of numpy / sympy arrays.
v [3 3 1 3 4]
w [2 1 1 2 2]
v@w 24
v.T@w 24
w.T@v 24
v.T@(A@B)@w 4523
w.T@(A@B).T@v 4523
(A@B).T [[61 53 64 39 33]
 [52 53 62 37 32]
 [42 40 48 25 23]
 [54 57 63 35 33]
 [41 33 42 29 21]]
B.T@A.T [[61 53 64 39 33]
 [52 53 62 37 32]
 [42 40 48 25 23]
 [54 57 63 35 33]
 [41 33 42 29 21]]
"""
```

```
import numpy as np

Arr = lambda txt: np.array([[eval(x.strip()) for x in y.split() if x.strip() !=
''] for y in txt.split(';') if y.split() != ''])

vs = Arr('1 1 1 1;1 -1 1 -1;1 1 -1 -1;1 -1 -1 1')

A = 1/16 * Arr('7 3 -13 7; 3 7 7 -13; -13 7 7 3; 7 -13 3 7')

eigs = np.linalg.eig(A)

# munge numpy-returned eigenvectors until they're scaled and ordered to match th
e provided vectors
eigvs = [x for x in zip(list(eigs[0]), eigs[1].T*2)]
eigvecs = np.vstack([eigvs[1][1], eigvs[0][1]*-1, eigvs[3][1]*-1, eigvs[2][1]*-1
]).T
eigvals = np.vstack([eigvs[1][0], eigvs[0][0], eigvs[3][0], eigvs[2][0]]).T
print('reordered and scaled eigenvectors of A', '\n', eigvecs)
print('provided vectors', '\n', vs)
print("eigenvals of A ordered according to provided vectors", eigvals)
print("eigenvals of A... scaled to match coefficients", eigvals/4)
print("coefficients provided:", [1/16, -1/4, +1/4, +3/8])


ANSWERS_a_b = """
reordered and scaled eigenvectors of A
[[ 1.  1.  1.  1.]
 [ 1. -1.  1. -1.]
 [ 1.  1. -1. -1.]
 [ 1. -1. -1.  1.]]
provided vectors
[[ 1  1  1  1]
 [ 1 -1  1 -1]
 [ 1  1 -1 -1]
 [ 1 -1 -1  1]]
eigenvals of A ordered according to provided vectors [[ 0.25 -1.    1.    1.5 ]]
eigenvals of A... scaled to match coefficients [[ 0.0625 -0.25    0.25    0.375 ]]
coefficients provided: [0.0625, -0.25, 0.25, 0.375]
"""

C = Arr('3 3 1 1; 3 3 1 1; 1 1 3 3; 1 1 3 3')/4
print("matrix provided as part C", "\n", C)

# helper function to build 2d matrix as a function of an index into `vs` defined
 above
matrix_from_index = lambda j: np.array([vs.T[j]]) * np.array([vs.T[j]]).T

print("matrix from 3rd vector, divided by 4", "\n", matrix_from_index(2)/4)

print("matrix from 3rd vector, divided by 4, summed with matrix from 1st vector, scaled by 2", "\n", matrix
_from_index(2)/4+matrix_from_index(0)/2)

ANSWERS_C = """
matrix provided as part C
[[ 0.75  0.75  0.25  0.25]
 [ 0.75  0.75  0.25  0.25]
 [ 0.25  0.25  0.75  0.75]
 [ 0.25  0.25  0.75  0.75]]
matrix from 3rd vector, divided by 4
```

```
[[ 0.25  0.25 −0.25 −0.25]
 [ 0.25  0.25 −0.25 −0.25]
 [−0.25 −0.25  0.25  0.25]
 [−0.25 −0.25  0.25  0.25]]
matrix from 3rd vector, divided by 4, summed with matrix from 1st vector, scaled by 2
[[ 0.75  0.75  0.25  0.25]
 [ 0.75  0.75  0.25  0.25]
 [ 0.25  0.25  0.75  0.75]
 [ 0.25  0.25  0.75  0.75]]
"""

D = Arr('0.5 0.4 0 0; 0.4 0.5 0 0; 0 0 0.5 0.1; 0 0 0.9 0.5')
eigs_D = np.linalg.eig(D)
print(eigs_D[1])

normalised_eigenvectors = np.vstack([eigs_D[1][i]/eigs_D[1][i][np.argmax(eigs_D[
1][i])] for i in range(eigs_D[1].shape[0])])

vs = normalised_eigenvectors

matrix_from_index = lambda j: np.array([vs.T[j]]) * np.array([vs.T[j]]).T

print('eigenvalues of part D', '\n', eigs_D[0])
print('the normalised eigenvectors of D', '\n', normalised_eigenvectors)
reconstructed = sum([(x*y) for (x,y) in zip(eigs_D[0], [matrix_from_index(i) for
 i in range(len(eigs_D[0]))])])/2
print('part D reconstructed as a sum of the product of the eigenvalues and the normalised eigenvectors', '\n',
reconstructed)

ANSWERS_DE = """
eigenvalues of matrix from part D
 [ 0.9  0.1  0.8  0.2]
the normalised eigenvectors of D
[[ 1. −1.  0.  0.]
 [ 1.  1.  0.  0.]
 [ 0. −0.  1. −1.]
 [ 0. −0.  1.  1.]]
part D reconstructed as a sum of the product of the eigenvalues and the normalised eigenvectors
[[ 0.5  0.4  0.   0. ]
 [ 0.4  0.5  0.   0. ]
 [ 0.   0.   0.5  0.3]
 [ 0.   0.   0.3  0.5]]
"""

ANSWER_F = """
    when we have a square, symmetric/diagonalisable(?) matrix with straight forward eigenvectors and eigenvalues, we
can find a combination of the normalised (?) eigenvectors extended to a 2dimensional pattern as V*V.T, given by the ei
genvalues associated with the vector, where the individual components summed gives us our original matrix. if the eige
nvectors are not orthogonal
    """
```

```
import numpy as np

Arr = lambda txt: np.array([[eval(x.strip()) for x in y.split() if x.strip() !=
''] for y in txt.split(';') if y.split() != ''])

vs = Arr('1 1 1 1;1 -1 1 -1;1 1 -1 -1;1 -1 -1 1')

A = 1/16 * Arr('7 3 -13 7; 3 7 7 -13; -13 7 7 3; 7 -13 3 7')

eigs = np.linalg.eig(A)

# munge numpy-returned eigenvectors until they're scaled and ordered to match th
e provided vectors
eigvs = [x for x in zip(list(eigs[0]), eigs[1].T*2)]
eigvecs = np.vstack([eigvs[1][1], eigvs[0][1]*-1, eigvs[3][1]*-1, eigvs[2][1]*-1
]).T
eigvals = np.vstack([eigvs[1][0], eigvs[0][0], eigvs[3][0], eigvs[2][0]]).T
print('reordered and scaled eigenvectors of A', '\n', eigvecs)
print('provided vectors', '\n', vs)
print("eigenvals of A ordered according to provided vectors", eigvals)
print("eigenvals of A... scaled to match coefficients", eigvals/4)
print("coefficients provided:", [1/16, -1/4, +1/4, +3/8])


ANSWERS_a_b = """
reordered and scaled eigenvectors of A
[[ 1.  1.  1.  1.]
 [ 1. -1.  1. -1.]
 [ 1.  1. -1. -1.]
 [ 1. -1. -1.  1.]]
provided vectors
[[ 1  1  1  1]
 [ 1 -1  1 -1]
 [ 1  1 -1 -1]
 [ 1 -1 -1  1]]
eigenvals of A ordered according to provided vectors [[ 0.25 -1.    1.    1.5 ]]
eigenvals of A... scaled to match coefficients [[ 0.0625 -0.25    0.25    0.375 ]]
coefficients provided: [0.0625, -0.25, 0.25, 0.375]
"""

C = Arr('3 3 1 1; 3 3 1 1; 1 1 3 3; 1 1 3 3')/4
print("matrix provided as part C", "\n", C)

# helper function to build 2d matrix as a function of an index into `vs` defined
 above
matrix_from_index = lambda j: np.array([vs.T[j]]) * np.array([vs.T[j]]).T

print("matrix from 3rd vector, divided by 4", "\n", matrix_from_index(2)/4)

print("matrix from 3rd vector, divided by 4, summed with matrix from 1st vector, scaled by 2", "\n", matrix
_from_index(2)/4+matrix_from_index(0)/2)

ANSWERS_C = """
matrix provided as part C
[[ 0.75  0.75  0.25  0.25]
 [ 0.75  0.75  0.25  0.25]
 [ 0.25  0.25  0.75  0.75]
 [ 0.25  0.25  0.75  0.75]]
matrix from 3rd vector, divided by 4
```

```
[[ 0.25  0.25 -0.25 -0.25]
 [ 0.25  0.25 -0.25 -0.25]
 [-0.25 -0.25  0.25  0.25]
 [-0.25 -0.25  0.25  0.25]]
matrix from 3rd vector, divided by 4, summed with matrix from 1st vector, scaled by 2
[[ 0.75  0.75  0.25  0.25]
 [ 0.75  0.75  0.25  0.25]
 [ 0.25  0.25  0.75  0.75]
 [ 0.25  0.25  0.75  0.75]]
"""

D = Arr('0.5 0.4 0 0; 0.4 0.5 0 0; 0 0 0.5 0.1; 0 0 0.9 0.5')
eigs_D = np.linalg.eig(D)
print(eigs_D[1])

normalised_eigenvectors = np.vstack([eigs_D[1][i]/eigs_D[1][i][np.argmax(eigs_D[
1][i])] for i in range(eigs_D[1].shape[0])])

vs = normalised_eigenvectors

matrix_from_index = lambda j: np.array([vs.T[j]]) * np.array([vs.T[j]]).T

print('eigenvalues of part D', '\n', eigs_D[0])
print('the normalised eigenvectors of D', '\n', normalised_eigenvectors)
reconstructed = sum([(x*y) for (x,y) in zip(eigs_D[0], [matrix_from_index(i) for
 i in range(len(eigs_D[0]))])])/2
print('part D reconstructed as a sum of the product of the eigenvalues and the normalised eigenvectors', '\n',
reconstructed)

ANSWERS_DE = """
eigenvalues of matrix from part D
 [ 0.9  0.1  0.8  0.2]
the normalised eigenvectors of D
[[ 1. -1.  0.  0.]
 [ 1.  1.  0.  0.]
 [ 0. -0.  1. -1.]
 [ 0. -0.  1.  1.]]
part D reconstructed as a sum of the product of the eigenvalues and the normalised eigenvectors
[[ 0.5  0.4  0.   0. ]
 [ 0.4  0.5  0.   0. ]
 [ 0.   0.   0.5  0.3]
 [ 0.   0.   0.3  0.5]]
"""

ANSWER_F = """
    when we have a square, symmetric/diagonalisable(?) matrix with straight forward eigenvectors and eigenvalues, we
can find a combination of the normalised (?) eigenvectors extended to a 2dimensional pattern as V*V.T, given by the ei
genvalues associated with the vector, where the individual components summed gives us our original matrix. if the eige
nvectors are not orthogonal
    """
```

```
import numpy as np

Arr = lambda txt: np.array([[eval(x.strip()) for x in y.split() if x.strip() !=
''] for y in txt.split(';') if y.split() != ''])

vs = Arr('1 1 1 1;1 -1 1 -1;1 1 -1 -1;1 -1 -1 1')

A = 1/16 * Arr('7 3 -13 7; 3 7 7 -13; -13 7 7 3; 7 -13 3 7')

eigs = np.linalg.eig(A)

# munge numpy-returned eigenvectors until they're scaled and ordered to match th
e provided vectors
eigvs = [x for x in zip(list(eigs[0]), eigs[1].T*2)]
eigvecs = np.vstack([eigvs[1][1], eigvs[0][1]*-1, eigvs[3][1]*-1, eigvs[2][1]*-1
]).T
eigvals = np.vstack([eigvs[1][0], eigvs[0][0], eigvs[3][0], eigvs[2][0]]).T
print('reordered and scaled eigenvectors of A', '\n', eigvecs)
print('provided vectors', '\n', vs)
print("eigenvals of A ordered according to provided vectors", eigvals)
print("eigenvals of A... scaled to match coefficients", eigvals/4)
print("coefficients provided:", [1/16, -1/4, +1/4, +3/8])


ANSWERS_a_b = """
reordered and scaled eigenvectors of A
[[ 1.  1.  1.  1.]
 [ 1. -1.  1. -1.]
 [ 1.  1. -1. -1.]
 [ 1. -1. -1.  1.]]
provided vectors
[[ 1  1  1  1]
 [ 1 -1  1 -1]
 [ 1  1 -1 -1]
 [ 1 -1 -1  1]]
eigenvals of A ordered according to provided vectors [[ 0.25 -1.    1.    1.5 ]]
eigenvals of A... scaled to match coefficients [[ 0.0625 -0.25    0.25    0.375 ]]
coefficients provided: [0.0625, -0.25, 0.25, 0.375]
"""

C = Arr('3 3 1 1; 3 3 1 1; 1 1 3 3; 1 1 3 3')/4
print("matrix provided as part C", "\n", C)

# helper function to build 2d matrix as a function of an index into `vs` defined
 above
matrix_from_index = lambda j: np.array([vs.T[j]]) * np.array([vs.T[j]]).T

print("matrix from 3rd vector, divided by 4", "\n", matrix_from_index(2)/4)

print("matrix from 3rd vector, divided by 4, summed with matrix from 1st vector, scaled by 2", "\n", matrix
_from_index(2)/4+matrix_from_index(0)/2)

ANSWERS_C = """
matrix provided as part C
[[ 0.75  0.75  0.25  0.25]
 [ 0.75  0.75  0.25  0.25]
 [ 0.25  0.25  0.75  0.75]
 [ 0.25  0.25  0.75  0.75]]
matrix from 3rd vector, divided by 4
```

```
[[ 0.25  0.25 -0.25 -0.25]
 [ 0.25  0.25 -0.25 -0.25]
 [-0.25 -0.25  0.25  0.25]
 [-0.25 -0.25  0.25  0.25]]
matrix from 3rd vector, divided by 4, summed with matrix from 1st vector, scaled by 2
[[ 0.75  0.75  0.25  0.25]
 [ 0.75  0.75  0.25  0.25]
 [ 0.25  0.25  0.75  0.75]
 [ 0.25  0.25  0.75  0.75]]
"""

D = Arr('0.5 0.4 0 0; 0.4 0.5 0 0; 0 0 0.5 0.1; 0 0 0.9 0.5')
eigs_D = np.linalg.eig(D)
print(eigs_D[1])

normalised_eigenvectors = np.vstack([eigs_D[1][i]/eigs_D[1][i][np.argmax(eigs_D[
1][i])] for i in range(eigs_D[1].shape[0])])

vs = normalised_eigenvectors

matrix_from_index = lambda j: np.array([vs.T[j]]) * np.array([vs.T[j]]).T

print('eigenvalues of part D', '\n', eigs_D[0])
print('the normalised eigenvectors of D', '\n', normalised_eigenvectors)
reconstructed = sum([(x*y) for (x,y) in zip(eigs_D[0], [matrix_from_index(i) for
 i in range(len(eigs_D[0]))])])/2
print('part D reconstructed as a sum of the product of the eigenvalues and the normalised eigenvectors', '\n',
reconstructed)

ANSWERS_DE = """
eigenvalues of matrix from part D
 [ 0.9  0.1  0.8  0.2]
the normalised eigenvectors of D
[[ 1. -1.  0.  0.]
 [ 1.  1.  0.  0.]
 [ 0. -0.  1. -1.]
 [ 0. -0.  1.  1.]]
part D reconstructed as a sum of the product of the eigenvalues and the normalised eigenvectors
[[ 0.5  0.4  0.   0. ]
 [ 0.4  0.5  0.   0. ]
 [ 0.   0.   0.5  0.3]
 [ 0.   0.   0.3  0.5]]
"""

ANSWER_F = """
    when we have a square, symmetric/diagonalisable(?) matrix with straight forward eigenvectors and eigenvalues, we
can find a combination of the normalised (?) eigenvectors extended to a 2dimensional pattern as V*V.T, given by the ei
genvalues associated with the vector, where the individual components summed gives us our original matrix. if the eige
nvectors are not orthogonal
    """
```

```
import numpy as np

Arr = lambda txt: np.array([[eval(x.strip()) for x in y.split() if x.strip() !=
''] for y in txt.split(';') if y.split() != ''])

vs = Arr('1 1 1 1;1 -1 1 -1;1 1 -1 -1;1 -1 -1 1')

A = 1/16 * Arr('7 3 -13 7; 3 7 7 -13; -13 7 7 3; 7 -13 3 7')

eigs = np.linalg.eig(A)

# munge numpy-returned eigenvectors until they're scaled and ordered to match th
e provided vectors
eigvs = [x for x in zip(list(eigs[0]), eigs[1].T*2)]
eigvecs = np.vstack([eigvs[1][1], eigvs[0][1]*-1, eigvs[3][1]*-1, eigvs[2][1]*-1
]).T
eigvals = np.vstack([eigvs[1][0], eigvs[0][0], eigvs[3][0], eigvs[2][0]]).T
print('reordered and scaled eigenvectors of A', '\n', eigvecs)
print('provided vectors', '\n', vs)
print("eigenvals of A ordered according to provided vectors", eigvals)
print("eigenvals of A... scaled to match coefficients", eigvals/4)
print("coefficients provided:", [1/16, -1/4, +1/4, +3/8])


ANSWERS_a_b = """
reordered and scaled eigenvectors of A
[[ 1.  1.  1.  1.]
 [ 1. -1.  1. -1.]
 [ 1.  1. -1. -1.]
 [ 1. -1. -1.  1.]]
provided vectors
[[ 1  1  1  1]
 [ 1 -1  1 -1]
 [ 1  1 -1 -1]
 [ 1 -1 -1  1]]
eigenvals of A ordered according to provided vectors [[ 0.25 -1.    1.    1.5 ]]
eigenvals of A... scaled to match coefficients [[ 0.0625 -0.25    0.25    0.375 ]]
coefficients provided: [0.0625, -0.25, 0.25, 0.375]
"""

C = Arr('3 3 1 1; 3 3 1 1; 1 1 3 3; 1 1 3 3')/4
print("matrix provided as part C", "\n", C)

# helper function to build 2d matrix as a function of an index into `vs` defined
 above
matrix_from_index = lambda j: np.array([vs.T[j]]) * np.array([vs.T[j]]).T

print("matrix from 3rd vector, divided by 4", "\n", matrix_from_index(2)/4)

print("matrix from 3rd vector, divided by 4, summed with matrix from 1st vector, scaled by 2", "\n", matrix
_from_index(2)/4+matrix_from_index(0)/2)

ANSWERS_C = """
matrix provided as part C
[[ 0.75  0.75  0.25  0.25]
 [ 0.75  0.75  0.25  0.25]
 [ 0.25  0.25  0.75  0.75]
 [ 0.25  0.25  0.75  0.75]]
matrix from 3rd vector, divided by 4
```

```
[[ 0.25  0.25 −0.25 −0.25]
 [ 0.25  0.25 −0.25 −0.25]
 [−0.25 −0.25  0.25  0.25]
 [−0.25 −0.25  0.25  0.25]]
matrix from 3rd vector, divided by 4, summed with matrix from 1st vector, scaled by 2
[[ 0.75  0.75  0.25  0.25]
 [ 0.75  0.75  0.25  0.25]
 [ 0.25  0.25  0.75  0.75]
 [ 0.25  0.25  0.75  0.75]]
"""

D = Arr('0.5 0.4 0 0; 0.4 0.5 0 0; 0 0 0.5 0.1; 0 0 0.9 0.5')
eigs_D = np.linalg.eig(D)
print(eigs_D[1])

normalised_eigenvectors = np.vstack([eigs_D[1][i]/eigs_D[1][i][np.argmax(eigs_D[
1][i])] for i in range(eigs_D[1].shape[0])])

vs = normalised_eigenvectors

matrix_from_index = lambda j: np.array([vs.T[j]]) * np.array([vs.T[j]]).T

print('eigenvalues of part D', '\n', eigs_D[0])
print('the normalised eigenvectors of D', '\n', normalised_eigenvectors)
reconstructed = sum([(x*y) for (x,y) in zip(eigs_D[0], [matrix_from_index(i) for
 i in range(len(eigs_D[0]))])])/2
print('part D reconstructed as a sum of the product of the eigenvalues and the normalised eigenvectors', '\n',
reconstructed)

ANSWERS_DE = """
eigenvalues of matrix from part D
 [ 0.9  0.1  0.8  0.2]
the normalised eigenvectors of D
[[ 1. −1.  0.  0.]
 [ 1.  1.  0.  0.]
 [ 0. −0.  1. −1.]
 [ 0. −0.  1.  1.]]
part D reconstructed as a sum of the product of the eigenvalues and the normalised eigenvectors
[[ 0.5  0.4  0.   0. ]
 [ 0.4  0.5  0.   0. ]
 [ 0.   0.   0.5  0.3]
 [ 0.   0.   0.3  0.5]]
"""

ANSWER_F = """
   when we have a square, symmetric/diagonalisable(?) matrix with straight forward eigenvectors and eigenvalues, we
can find a combination of the normalised (?) eigenvectors extended to a 2dimensional pattern as V*V.T, given by the ei
genvalues associated with the vector, where the individual components summed gives us our original matrix. if the eige
nvectors are not orthogonal
   """
```

```
import numpy as np

Arr = lambda txt: np.array([[eval(x.strip()) for x in y.split() if x.strip() !=
''] for y in txt.split(';') if y.split() != ''])

A = (1/3)*Arr('2 1 −2; 2 −2 1; 1 2 2')
print('A', A)
print('inverse A', np.linalg.inv(A))

B = (1/5)*Arr('0 0 3 4; 0 0 −4 3; 4 3 0 0; 4 −3 0 0')
print('B', B)
print('inverse B', np.linalg.inv(B))

print("the inverse of a matrix is equal to the transposition of the mutually orthogonal vectors that when dotted with
a vector of constants is equal to.... ugh.")

ANSWERS = """
A [[ 0.66666667  0.33333333 −0.66666667]
 [ 0.66666667 −0.66666667  0.33333333]
 [ 0.33333333  0.66666667  0.66666667]]
inverse A [[ 0.66666667  0.66666667  0.33333333]
 [ 0.33333333 −0.66666667  0.66666667]
 [−0.66666667  0.33333333  0.66666667]]
B [[ 0.   0.   0.6  0.8]
 [ 0.   0.  −0.8  0.6]
 [ 0.8  0.6  0.   0. ]
 [ 0.8 −0.6  0.   0. ]]
inverse B [[ 0.          0.          0.625       0.625     ]
 [ 0.          0.          0.83333333 −0.83333333]
 [ 0.6        −0.8         0.          0.        ]
 [ 0.8         0.6         0.          0.        ]]
the inverse of a matrix is equal to the transposition of the mutually orthogonal vectors that when dotted with a vector of
 constants is equal to.... ugh.
"""
```

```
import numpy as np

Arr = lambda txt: np.array([[eval(x.strip()) for x in y.split() if x.strip() !=
''] for y in txt.split(';') if y.split() != ''])

A = (1/3)*Arr('2 1 -2; 2 -2 1; 1 2 2')
print('A', A)
print('inverse A', np.linalg.inv(A))

B = (1/5)*Arr('0 0 3 4; 0 0 -4 3; 4 3 0 0; 4 -3 0 0')
print('B', B)
print('inverse B', np.linalg.inv(B))

print("the inverse of a matrix is equal to the transposition of the mutually orthogonal vectors that when dotted with
a vector of constants is equal to.... ugh.")

ANSWERS = """
A [[ 0.66666667  0.33333333 -0.66666667]
 [ 0.66666667 -0.66666667  0.33333333]
 [ 0.33333333  0.66666667  0.66666667]]
inverse A [[ 0.66666667  0.66666667  0.33333333]
 [ 0.33333333 -0.66666667  0.66666667]
 [-0.66666667  0.33333333  0.66666667]]
B [[ 0.   0.   0.6  0.8]
 [ 0.   0.  -0.8  0.6]
 [ 0.8  0.6  0.   0. ]
 [ 0.8 -0.6  0.   0. ]]
inverse B [[ 0.          0.          0.625       0.625     ]
 [ 0.          0.          0.83333333 -0.83333333]
 [ 0.6        -0.8         0.          0.        ]
 [ 0.8         0.6         0.          0.        ]]
the inverse of a matrix is equal to the transposition of the mutually orthogonal vectors that when dotted with a vector of
 constants is equal to.... ugh.
"""
```

```
import numpy as np

Arr = lambda txt: np.array([[eval(x.strip()) for x in y.split() if x.strip() != ''] for y in txt.split(';') if y.split() != ''])

A = (1/3)*Arr('2 1 −2; 2 −2 1; 1 2 2')
print('A', A)
print('inverse A', np.linalg.inv(A))

B = (1/5)*Arr('0 0 3 4; 0 0 −4 3; 4 3 0 0; 4 −3 0 0')
print('B', B)
print('inverse B', np.linalg.inv(B))

print("the inverse of a matrix is equal to the transposition of the mutually orthogonal vectors that when dotted with a vector of constants is equal to.... ugh.")

ANSWERS = """
A [[ 0.66666667  0.33333333 −0.66666667]
 [ 0.66666667 −0.66666667  0.33333333]
 [ 0.33333333  0.66666667  0.66666667]]
inverse A [[ 0.66666667  0.66666667  0.33333333]
 [ 0.33333333 −0.66666667  0.66666667]
 [−0.66666667  0.33333333  0.66666667]]
B [[ 0.   0.   0.6  0.8]
 [ 0.   0.  −0.8  0.6]
 [ 0.8  0.6  0.   0. ]
 [ 0.8 −0.6  0.   0. ]]
inverse B [[ 0.          0.          0.625       0.625     ]
 [ 0.          0.          0.83333333 −0.83333333]
 [ 0.6        −0.8         0.          0.        ]
 [ 0.8         0.6         0.          0.        ]]
the inverse of a matrix is equal to the transposition of the mutually orthogonal vectors that when dotted with a vector of
 constants is equal to.... ugh.
"""
```

```
import numpy as np

Arr = lambda txt: np.array([[eval(x.strip()) for x in y.split() if x.strip() != ''] for y in txt.split(';') if y.split() != ''])

A = (1/3)*Arr('2 1 -2; 2 -2 1; 1 2 2')
print('A', A)
print('inverse A', np.linalg.inv(A))

B = (1/5)*Arr('0 0 3 4; 0 0 -4 3; 4 3 0 0; 4 -3 0 0')
print('B', B)
print('inverse B', np.linalg.inv(B))

print("the inverse of a matrix is equal to the transposition of the mutually orthogonal vectors that when dotted with a vector of constants is equal to.... ugh.")

ANSWERS = """
A [[ 0.66666667  0.33333333 -0.66666667]
 [ 0.66666667 -0.66666667  0.33333333]
 [ 0.33333333  0.66666667  0.66666667]]
inverse A [[ 0.66666667  0.66666667  0.33333333]
 [ 0.33333333 -0.66666667  0.66666667]
 [-0.66666667  0.33333333  0.66666667]]
B [[ 0.   0.   0.6  0.8]
 [ 0.   0.  -0.8  0.6]
 [ 0.8  0.6  0.   0. ]
 [ 0.8 -0.6  0.   0. ]]
inverse B [[ 0.          0.          0.625       0.625     ]
 [ 0.          0.          0.83333333 -0.83333333]
 [ 0.6        -0.8         0.          0.        ]
 [ 0.8         0.6         0.          0.        ]]
the inverse of a matrix is equal to the transposition of the mutually orthogonal vectors that when dotted with a vector of
 constants is equal to.... ugh.
"""
```