Solution 3.3 3H The following program meets the requirements:

```python
"""Solution to Chapter 3 problem 3H"""
import numpy as np

# %% [markdown]
## Introduction
# This program defines matrices A and B and vectors x and y. It then
# computes several quantities that require matrix operations.
# %% [markdown]
## a. Define Arrays
# %%
A = np.array(
    [
        [2, 1, 9, 0],
        [0, -1, -2, 3],
        [-3, 0, 8, -4],
    ]
)
B = np.array(
    [
        [0, 9, -1],
        [1, 0, 3],
        [0, -1, 1],
    ]
)
x = np.array([[1], [0], [-1]])
y = np.array([[3, 0, -1]])
# %% [markdown]
## b. Compute and Print Quantitites

### i. $BA$
# %%
print("B A = \n", B @ A)
# %% [markdown]
### ii. $A^\top B - 6 J_{4 \times 3}$
# %%
print("A.T B - 6 J_4x3 = \n", A.T @ B - 6 * np.ones((4, 3)))
# %% [markdown]
### iii. $B x + y^\top$
# %%
print("B x + y.T = \n", B @ x + y.T)
# %% [markdown]
### iv. $x y + B$
# %%
print("x y + B = \n", x @ y + B)
# %% [markdown]
```

```
### v. yx
# %%
print("y x = \n", y @ x)
# %% [markdown]
### vi. yB^{-1}x
# %%
print("y B^-1 x = \n", y @ np.linalg.inv(B) @ x)
# %% [markdown]
### vii. CB
# %%
C = A[:, :3]  # First 3 columns of A (view)
print("C B = \n", C @ B)

# %% tags=["active-py"]
import sys

sys.path.append("../")
import engcom.engcom as engcom

pub = engcom.Publication(title="Problem 3H", author="Rico Picone")
pub.write(to="md")
```

This program prints the following to the console:

```
B A =
 [[  3  -9 -26  31]
 [ -7   1  33 -12]
 [ -3   1  10  -7]]
A.T B - 6 J_4x3 =
 [[ -6.  15. -11.]
 [ -7.   3. -10.]
 [ -8.  67. -13.]
 [ -3.  -2.  -1.]]
B x + y.T =
 [[ 4]
 [-2]
 [-2]]
x y + B =
 [[ 3  9 -2]
 [ 1  0  3]
 [-3 -1  2]]
y x =
 [[4]]
y B^-1 x =
 [[10.]]
C B =
[[  1   9  10]
 [ -1   2  -5]
 [  0 -35  11]]
```

Problem 3.4 DI Consider the array:

```
a = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8], [9, 10, 11]])  # 4x3
```

Write a program that performs and prints the results of the following operations on a *without* using `for` loops:

a. Adds 1 to all elements
b. Adds 1 to the last column
c. Flattens a to a vector
d. Reshapes a into a 3 × 4 matrix
e. Adds the vector [1, 2, 3] to each row
f. Adds the vector [1, 2, 3, 4] to each column
g. Reshapes a to a column vector
h. Reshapes a to a row vector

Solution 3.4 DI The following program meets the requirements:

```
""""Solution to Chapter 3 problem DI"""
import numpy as np

# %% [markdown]
## Introduction
# This program defines a NumPy array and prints the results of
# several operations.
# %% [markdown]
## Define Array
# %%
a = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8], [9, 10, 11]])  # 4x3
# %% [markdown]
## a. Adds 1 to all elements
print("Adds 1 to all elements:\n", a + 1)
# %% [markdown]
## b. Adds 1 to the last column
b = a.copy()
b[:, -1] = b[:, -1] + 1
print("Adds 1 to the last column:\n", b)
# %% [markdown]
## c. Flattens a to a vector
print("Flattens a to a vector\n", a.flatten())
# %% [markdown]
## d. Reshapes a into a 3×4 matrix
```

```python
print("Reshapes a into a 3×4 matrix:\n", a.reshape((3, 4)))
# %% [markdown]
## e. Adds the vector [1, 2, 3] to each row
c = np.array([1, 2, 3])
cr = c.reshape((1, 3))  # Reshape for broadcasting
print("Adds the vector [1, 2, 3] to each row:\n", a + cr)
# %% [markdown]
## f. Adds the vector [1, 2, 3, 4] to each column
d = np.array([1, 2, 3, 4])
dr = d.reshape((4, 1))  # Reshape for broadcasting
print("Adds the vector [1, 2, 3, 4] to each column:\n", a + dr)
# %% [markdown]
## g. Reshapes a to a column vector
print("Reshapes a to a column vector:\n", a.reshape((a.size, 1)))
# %% [markdown]
## h. Reshapes a to a row vector
print("Reshapes a to a row vector:\n", a.reshape((1, a.size)))
```

This program prints the following to the console:

```
Adds 1 to all elements:
 [[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
Adds 1 to the last column:
 [[ 0  1  3]
 [ 3  4  6]
 [ 6  7  9]
 [ 9 10 12]]
Flattens a to a vector
 [ 0  1  2  3  4  5  6  7  8  9 10 11]
Reshapes a into a 3×4 matrix:
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
Adds the vector [1, 2, 3] to each row:
 [[ 1  3  5]
 [ 4  6  8]
 [ 7  9 11]
 [10 12 14]]
Adds the vector [1, 2, 3, 4] to each column:
 [[ 1  2  3]
 [ 5  6  7]
 [ 9 10 11]
 [13 14 15]]
Reshapes a to a column vector:
 [[ 0]
 [ 1]
 [ 2]
 [ 3]
 [ 4]
 [ 5]
 [ 6]
 [ 7]
 [ 8]
 [ 9]
 [10]
 [11]]
Reshapes a to a row vector:
 [[ 0  1  2  3  4  5  6  7  8  9 10 11]]
```

Problem 3.5 QX Write vectorized Python functions that operate element-wise on array arguments for the following mathematical functions:

a. $f(x) = x^2 + 3x + 9$
b. $g(x) = 1 + \sin^2 x$
c. $h(x, y) = e^{-3x} + \ln y$
d. $F(x, y) = \lfloor x/y \rfloor$
e. $G(x, y) = \begin{cases} x^2 + y^2 & \text{if } x > y \\ 2x & \text{otherwise} \end{cases}$

Solution 3.5 QX The following program meets the requirements:

```
""""Solution to Chapter 3 problem QX"""
import numpy as np

# %% [markdown]
## Introduction
# This program defines several mathematical functions as vectorized
# functions that can handle NumPy array inputs.
# %% [markdown]
## a. $f(x) = x^2 + 3x + 9$
# %%
def f(x: np.ndarray) -> np.ndarray:
    return x**2 + 3 * x + 9


# %% [markdown]
## b. $g(x) = 1 + \sin^2 x$
# %%
def g(x: np.ndarray) -> np.ndarray:
    return 1 + np.sin(x) ** 2


# %% [markdown]
## c. $h(x, y) = e^{-3x} + \ln y$
# %%
def h(x: np.ndarray, y: np.ndarray) -> np.ndarray:
    return np.exp(-3 * x) + np.log(y)


# %% [markdown]
## d. $F(x, y) = \lfloor x/y \rfloor$
# %%
def F(x: np.ndarray, y: np.ndarray) -> np.ndarray:
    return np.floor(x / y)


# %% [markdown]
## e. $G(x, y) = \begin{cases} x^2 + y^2 \& \text{if } x > y \\ 2x \& \text{otherwise} \end{cases}$
# %%
def G(x: np.ndarray, y: np.ndarray) -> np.ndarray:
    return np.where(x > y, x**2 + y**2, 2 * x)


# %% [markdown]
## Call Functions and Print
# %%
functions_args = (
    (f, 1),
    (g, 1),
    (h, 2),
    (F, 2),
    (G, 2),
)  # (fun, nargs)
x = np.array([1, 5, 10, 20, 30])
y = np.array([2, 7, 5, 10, 30])
print(f"x = {x}\ny = {y}")

for function_args in functions_args:
    if function_args[1] == 1:
        printable = np.array2string(function_args[0](x), precision=3)
        print(f"{function_args[0].__name__}(x) =", printable)
    elif function_args[1] == 2:
        printable = np.array2string(function_args[0](x, y), precision=3)
        print(f"{function_args[0].__name__}(x, y) =", printable)
```

This program prints the following to the console:

```
x = [ 1  5 10 20 30]
y = [ 2  7  5 10 30]
f(x) = [ 13  49 139 469 999]
g(x) = [1.708 1.92  1.296 1.833 1.976]
h(x, y) = [0.743 1.946 1.609 2.303 3.401]
F(x, y) = [0. 0. 2. 2. 1.]
G(x, y) = [  2  10 125 500  60]
```