week_6

February 19, 2025

1 Week 6

1.1 TA Solution

1.1.1 Abhisek Sarkar

as20ms091@iiserkol.ac.in

1.1.2 Q1.

Input (using input()) a list of alternating names and ages e.g. [‘kripa’, ‘37’, ‘arun’, ‘45’, ‘dipa’, ‘40’] and create a dictionary such the (i, i+1)th elements (i = 0, 2, …) form the key value pairs. For the mentioned example, the dictionary will be {‘kripa’: ‘37’, ‘arun’: ‘45’, ‘dipa’: ‘40’}

[2]:
```
# Take input from the user
input_list = input("Enter a list of names and ages (comma-separated): ").
↪split(',')

# Strip any extra whitespace
input_list = [item.strip() for item in input_list]

# Convert the list into a dictionary using pairs of elements
name_age_dict = {input_list[i]: input_list[i+1] for i in range(0,␣
↪len(input_list), 2)}

# Print the dictionary
print(name_age_dict)
```

```
{'kripa': '37', 'arun': '45', 'dipa': '40'}
```

1.1.3 Q2.

For the same input as Q1, create a list of tuples of the (i, i+1)th elements (i = 0, 2, …) of the input list. For the mentioned example, the output list will be [(‘kripa’, ‘37’), (‘arun’, ‘32’), (‘dipa’, ’40)]

[3]:
```
# Take input from the user
#input_list = input("Enter a list of names and ages (comma-separated): ").
↪split(',')

# Strip any extra whitespace
```

```
input_list = [item.strip() for item in input_list]

# Convert the list into a list of tuples using pairs of elements
tuple_list = [(input_list[i], input_list[i+1]) for i in range(0,␣
↪len(input_list), 2)]

# Print the list of tuples
print(tuple_list)
```

```
[('kripa', '37'), ('arun', '45'), ('dipa', '40')]
```

1.1.4 Q3

Use numpy linalg.solve to find the point of intersection of the three planes x + 2y + 3z = 2, 4x + 8y +66z = 3 and 7x + 81y + 9z = 4. Perform the said operation if the coefficient determinant is non-singular. This can be checked by the function np.linalg.det(). You may consider a determinant to be singular if np.linalg.det() is close to zero (say abs() value less than 0.00001).

[4]:
```
import numpy as np

# Coefficient matrix
A = np.array([
    [1, 2, 3],
    [4, 8, 66],
    [7, 81, 9]
])

# Right-hand side constants
b = np.array([2, 3, 4])

# Check if the determinant is non-singular
det_A = np.linalg.det(A)

if abs(det_A) < 0.00001:
    print("The coefficient matrix is singular or nearly singular. No unique␣
↪solution exists.")
else:
    # Solve the system of equations
    solution = np.linalg.solve(A, b)
    print("Point of intersection:", solution)
```

```
Point of intersection: [ 2.60945274 -0.16583748 -0.09259259]
```

1.1.5 Q4

Write a function d2b() that takes a decimal number as argument and returns its binary equivalent. Write a numpy ufunc function numpy_DecimalToBinary that uses d2b() and applies the same operation elementwise on a numpy array A. E.g. if A is 1, 2, 3, 4, 5, numpy_DecimalToBinary(A) will produce 1 10 11 100 101. Note that all the elements of numpy_DecimalToBinary(A) should be of 'int' type and not 'str' type.

[5]:
```python
import numpy as np

# Function to convert a decimal number to binary (returns an integer)
def d2b(n):
    return int(bin(n)[2:])  # Convert to binary string and then to integer

# Create a numpy universal function (ufunc)
numpy_DecimalToBinary = np.vectorize(d2b)

# Example usage
A = np.array([1, 2, 3, 4, 5])
binary_array = numpy_DecimalToBinary(A)
print(binary_array)
```

```
[  1  10  11 100 101]
```

1.1.6 Q5

Consider a numpy 2-D array of the form [[50, 60, 70], [67, 88, 90], [60, 78, 97]] where the ith 1-D array contains the marks of the ith student in three subjects (in this order. E.g. 50, 60, 70 are the subject1, subject2, subject3 marks respectively of student-0. Use numpy sum function only to i) create a 1-D numpy array with the sum of the marks of individual students ii) create a 1-D numpy array with the sum of subject-wise marks. Also, do these operations to produce 2-D numpy arrays with the same content.

[6]:
```python
import numpy as np

# Define the 2D NumPy array
marks = np.array([
    [50, 60, 70],
    [67, 88, 90],
    [60, 78, 97]
])

# i) Sum of marks for each student (row-wise sum)
student_sums_1D = np.sum(marks, axis=1)  # 1D array
student_sums_2D = student_sums_1D.reshape(-1, 1)  # Convert to 2D column vector

# ii) Sum of marks for each subject (column-wise sum)
subject_sums_1D = np.sum(marks, axis=0)  # 1D array
subject_sums_2D = subject_sums_1D.reshape(1, -1)  # Convert to 2D row vector

# Print results
print("1D Array - Student-wise Sum:", student_sums_1D)
print("2D Array - Student-wise Sum:\n", student_sums_2D)
```

```
print("1D Array - Subject-wise Sum:", subject_sums_1D)
print("2D Array - Subject-wise Sum:\n", subject_sums_2D)
```

```
1D Array - Student-wise Sum: [180 245 235]
2D Array - Student-wise Sum:
 [[180]
 [245]
 [235]]
1D Array - Subject-wise Sum: [177 226 257]
2D Array - Subject-wise Sum:
 [[177 226 257]]
```

1.1.7 Q6

Given two 1-D numpy arrays A and B, remove the elements in A which are also in B and store the resulting array in C. Use numpy set operations.

[7]:
```
import numpy as np

# Example input arrays
A = np.array([1, 2, 3, 4, 5, 6])
B = np.array([2, 4, 6])

# Remove elements in A that are also in B
C = np.setdiff1d(A, B)

# Print the result
print("Resulting array C:", C)
```

```
Resulting array C: [1 3 5]
```

1.1.8 Q7

Given two numpy 2-D arrays arr1 = np.array([[1, 2], [4, 5]]), arr2 = np.array([[3, 3], [1,1]]) explore the difference between np.multiply(arr1, arr2) and np.matmul(arr1, arr2).

[8]:
```
import numpy as np

# Define the 2D arrays
arr1 = np.array([[1, 2], [4, 5]])
arr2 = np.array([[3, 3], [1, 1]])

# Element-wise multiplication
elementwise_product = np.multiply(arr1, arr2)

# Matrix multiplication (dot product)
matrix_product = np.matmul(arr1, arr2)

# Print results
```

```
print("Element-wise multiplication (np.multiply):\n", elementwise_product)
print("\nMatrix multiplication (np.matmul):\n", matrix_product)
```

```
Element-wise multiplication (np.multiply):
 [[3 6]
 [4 5]]

Matrix multiplication (np.matmul):
 [[ 5  5]
 [17 17]]
```

1.1.9 Q8

Consider a python list of numbers L and a function f(x) = x3 + 1. Apply f(x) on every element of L and store the result in another list Lout. Now, create a numpy array A from L and apply f(x) element-wise on A and store the result in another numpy array Aout.

[9]:
```
import numpy as np

# Define the function f(x) = x^3 + 1
def f(x):
    return x**3 + 1

# Python list approach
L = [1, 2, 3, 4, 5]  # Example list
Lout = [f(x) for x in L]  # Apply f(x) to each element

# NumPy array approach
A = np.array(L)  # Convert list to NumPy array
Aout = f(A)  # Apply f(x) element-wise using NumPy

# Print results
print("List output:", Lout)
print("NumPy output:", Aout)
```

```
List output: [2, 9, 28, 65, 126]
NumPy output: [  2   9  28  65 126]
```

1.1.10 Q9

Given a numpy array X that can contain any integer (negative, positive or zero) and consider a function f(x) defined as follows: f returns 0 if x < 0, returns 1 if x is 0 and returns x%3 otherwise. Apply f(x) elementwise on X. [Hint: use frompyfunc]. Also apply vectorize, to solve the same problem [same as frompyfunc except that the number of arguments are not passed in vectorize].

[10]:
```
import numpy as np

# Define the function f(x)
def f(x):
```

```
    if x < 0:
        return 0
    elif x == 0:
        return 1
    else:
        return x % 3

# Create a sample NumPy array
X = np.array([-5, 0, 3, 7, -2, 9, 12, -1])

# Apply using np.frompyfunc()
f_frompyfunc = np.frompyfunc(f, 1, 1)  # 1 input, 1 output
X_out_frompyfunc = f_frompyfunc(X)

# Apply using np.vectorize()
f_vectorized = np.vectorize(f)
X_out_vectorized = f_vectorized(X)

# Print results
print("Original array:", X)
print("Output using frompyfunc:", X_out_frompyfunc)
print("Output using vectorize:", X_out_vectorized)
```

```
Original array: [-5  0  3  7 -2  9 12 -1]
Output using frompyfunc: [0 1 0 1 0 0 0 0]
Output using vectorize: [0 1 0 1 0 0 0 0]
```

1.1.11 Q10

Use linspace in numpy to generate 10 equispaced points in the interval [20, 21] (a) endpoint 21 is included, (b) endpoint 21 is not included (c) return the interval length for case (a)

[11]:
```
import numpy as np

# (a) Including the endpoint 21
points_inclusive = np.linspace(20, 21, num=10, endpoint=True)

# (b) Excluding the endpoint 21
points_exclusive = np.linspace(20, 21, num=10, endpoint=False)

# (c) Interval length for case (a)
interval_length = points_inclusive[1] - points_inclusive[0]

# Print results
print("10 points (endpoint included):", points_inclusive)
print("10 points (endpoint NOT included):", points_exclusive)
print("Interval length for case (a):", interval_length)
```

```
10 points (endpoint included): [20.         20.11111111 20.22222222 20.33333333
20.44444444 20.55555556
 20.66666667 20.77777778 20.88888889 21.        ]
10 points (endpoint NOT included): [20.  20.1 20.2 20.3 20.4 20.5 20.6 20.7 20.8
20.9]
Interval length for case (a): 0.11111111111111072
```