```
{
 "cells": [
  {
   "cell_type": "raw",
   "id": "7872015e-f668-459a-b78d-cff5f525cc96",
   "metadata": {},
   "source": [
    "1. Explain the purpose and advantages of NumPy in scientific computing and data analysis. How does it\n",
    "enhance Python's capabilities for numerical operations?\n",
    "\n",
    "ans.the purpose and advantages of numpy in scientific computing and data analysis is we can do many mathematicle operations by \n",
    "the help of this library.we can all library functions.we can efficiantly handle array and other operatons."
   ]
  },
  {
   "cell_type": "raw",
   "id": "712a2994-2582-4546-9711-4547b16ef00e",
   "metadata": {},
   "source": [
    "2. Compare and contrast np.mean() and np.average() functions in NumPy. When would you use one over the\n",
    "other?\n",
    "\n",
    "ans.np.mean()\n",
    "\n",
    "Use to calculate arithmetic mean\n",
    "\n",
    "\n",
    "\n",
    "All elements have equal weight\n",
    "\n",
    "A\n",
    "\n",
    "Weight cannot be passed trough the parameter of the given function.\n",
    "\n",
    "\n",
    "\n",
    "Syntax :\n",
    "\n",
    "np.mean(arr, axis = None)\n",
    "\n",
    "where 'arr' is the given array.\n",
    "\n",
    "np.average()\n",
    "\n",
    "\n",
    "Use to calculate the arithmetic mean as well as weighted average.\n",
    "\n",
    "\n",
    "All elements may or may not have equal weight.\n",
    "\n",
    "\n",
    "\n",
    "Weight can be passed through the parameter of the given function.\n",
    "\n",
    "\n",
    "Syntax :\n",
    "\n",
    "numpy.average(arr, axis = None, weights = None)\n",
    "\n",
    "Where 'arr' is the given array"
   ]
  },
  {
   "cell_type": "raw",
   "id": "fb11d9d4-71d1-4a1a-a809-c4752e49eec7",
   "metadata": {},
   "source": [
    "3. Describe the methods for reversing a NumPy array along different axes. Provide examples for 1D and 2D\n",
    "arrays.\n",
    "ans.we can do this by using flip functions in numpy we can reverse the array\n"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 3,
   "id": "f47586ec-c0a7-4e4b-b495-969f0daf5724",
   "metadata": {},
   "outputs": [
    {
     "data": {
      "text/plain": [
       "array([ 66, 657,   5,  44,   3,   2])"
      ]
     },
     "execution_count": 3,
     "metadata": {},
     "output_type": "execute_result"
    }
   ],
   "source": [
    "import numpy as np\n",
    "import pandas as pd\n",
    "\n",
    "\n",
    "#using array\n",
    "arr=[2,3,44,5,657,66]\n",
    "#using flip\n",
    "reverse_arr=np.flip(arr)\n",
    "reverse_arr"
   ]
  },
  {
   "cell_type": "raw",
   "id": "59567662-e4b6-4f7c-b99a-8b47a8c74a7a",
   "metadata": {},
   "source": [
```

```
  "4. How can you determine the data type of elements in a NumPy array? Discuss the importance of data types\n",
  "in memory management and performance.\n",
  "ans.we can determine the datatype in a numpy array that by the help of dtype functions,size\n",
  "data type importance\n",
  "size define\n",
  "Every ndarray has an associated data type (dtype) object. This data type object (dtype) informs us about the layout of the array. This means it gives us i
  "\n",
  "Type of the data (integer, float, Python object etc.)\n",
  "Size of the data (number of bytes)\n",
  "Byte order of the data (little-endian or big-endian)\n",
  "If the data type is a sub-array, what is its shape and data type."
 ]
},
{
 "cell_type": "raw",
 "id": "d0b8ca6a-9983-45f0-94d8-417246131d77",
 "metadata": {},
 "source": [
  "5. Define ndarrays in NumPy and explain their key features. How do they differ from standard Python lists?\n",
  "Key Features of NumPy ndarrays:\n",
  "Homogeneous: All elements in an ndarray are of the same data type (e.g., integers, floats). This uniformity allows for faster computations.\n",
  "\n",
  "N-dimensional: They can have multiple dimensions, from 1D arrays (vectors) to 2D arrays (matrices), and even higher dimensions (e.g., 3D for tensors).\n",
  "\n",
  "Efficient Memory Usage: Unlike Python lists, which are general-purpose containers and can hold objects of different types, NumPy ndarrays use contiguous b
  "\n",
  "Vectorized Operations: NumPy provides a large set of operations that can be performed on arrays without the need for explicit loops, making computations f
  "\n",
  "Mathematical Functions: NumPy provides built-in support for mathematical operations like matrix multiplication, linear algebra, and statistical functions.
  "\n",
  "Indexing and Slicing: ndarrays support advanced indexing, slicing, and broadcasting, allowing for more efficient data manipulation and transformation.\n",
  "\n",
  "Shape and Dimensions: Each ndarray has a shape and a dimension count (rank). The shape defines the size of each dimension, and the number of dimensions ca
  "\n",
  "How ndarrays Differ from Python Lists:\n",
  "Performance: ndarrays are much faster than Python lists for numerical computations, especially when working with large datasets. This is due to NumPy's us
  "\n",
  "Data Type Consistency: ndarrays can only store elements of the same type, while Python lists can hold heterogeneous data types (e.g., integers, strings, a
  "\n",
  "Dimensionality: Lists are inherently 1D, though they can contain other lists (nested lists) to represent multi-dimensional data. However, this is cumberso
  "\n",
  "Memory Efficiency: ndarrays use less memory than Python lists because they store data more compactly (i.e., with less overhead per element).\n",
  "\n",
  "Element-wise Operations: Python lists require explicit loops to perform operations on each element, whereas ndarrays support element-wise operations direc
  "\n"
 ]
},
{
 "cell_type": "code",
 "execution_count": 6,
 "id": "9d19a6f0-3929-4c32-b3b7-58e6a54ec89f",
 "metadata": {},
 "outputs": [],
 "source": [
  "\n",
  "\n",
  "# Creating a 1D ndarray\n",
  "arr = np.array([1, 2, 3, 4])\n",
  "\n",
  "# Creating a 2D ndarray\n",
  "arr_2d = np.array([[1, 2, 3], [4, 5, 6]])\n",
  "\n",
  "# Example of vectorized operations\n",
  "arr_squared = arr ** 2  # Squaring each element\n",
  "\n",
  "# Slicing and indexing in a 2D array\n",
  "element = arr_2d[1, 2]  # Accessing row 2, column 3 (value: 6)"
 ]
},
{
 "cell_type": "raw",
 "id": "26b407af-fe4b-4ce6-a9e8-0fba37cfbd63",
 "metadata": {},
 "source": [
  "6. Analyze the performance benefits of NumPy arrays over Python lists for large-scale numerical operations.\n",
  "\n",
  "1. Memory Efficiency\n",
  "NumPy arrays are stored in contiguous blocks of memory and use a fixed data type for all elements, allowing them to be more compact and require less memor
  "Python lists, on the other hand, are collections of references to objects, which means they use more memory per element (the reference and the object itse
  "Example:\n",
  "\n",
  "A NumPy array of integers will only store the raw integer data in memory, whereas a Python list stores references to integer objects, each of which has it
  "2. Vectorization and Avoiding Loops\n",
  "NumPy arrays support vectorized operations, allowing you to perform mathematical operations on entire arrays without explicit loops. These operations are
 ]
},
{
 "cell_type": "code",
 "execution_count": 7,
 "id": "158762f4-ebb6-426a-a274-ebc15a7e8b5b",
 "metadata": {},
 "outputs": [],
 "source": [
  "import numpy as np\n",
  "\n",
  "# With NumPy arrays\n",
  "arr = np.array([1, 2, 3, 4, 5])\n",
  "result = arr * 2  # Multiply each element by 2 (vectorized)\n",
  "\n",
  "# With Python lists\n",
  "py_list = [1, 2, 3, 4, 5]\n",
  "result_list = [x * 2 for x in py_list]  # Manual loop\n"
 ]
},
{
 "cell_type": "code",
```

```
   "execution_count": null,
   "id": "72b93236-523a-4584-be86-493c29d0b721",
   "metadata": {},
   "outputs": [],
   "source": []
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "7a46aab6-acce-4f00-a46c-b73086f4b69a",
   "metadata": {},
   "outputs": [],
   "source": []
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "00e32dc0-3af2-4193-8afe-16ddcf5b9b59",
   "metadata": {},
   "outputs": [],
   "source": []
  }
 ],
 "metadata": {
  "kernelspec": {
   "display_name": "Python 3 (ipykernel)",
   "language": "python",
   "name": "python3"
  },
  "language_info": {
   "codemirror_mode": {
    "name": "ipython",
    "version": 3
   },
   "file_extension": ".py",
   "mimetype": "text/x-python",
   "name": "python",
   "nbconvert_exporter": "python",
   "pygments_lexer": "ipython3",
   "version": "3.12.4"
  }
 },
 "nbformat": 4,
 "nbformat_minor": 5
}
```