Day_27_201123

January 23, 2024

1 NumPy

Numerical Python

- It is used for complex calculation
- It is faster than array
- We can perform calculation on 2D, 3D, 4D ……, nDm

Initialize the NumPy and Importing NumPy

[50]:
```
# Installing NumPy in system
!pip install numpy
import numpy as np # Importing Numpy
```

```
Requirement already satisfied: numpy in c:\data\env\lib\site-packages (1.26.1)
```

- How likely do you suggest our product to your friends and family? What would be your preference in this scenario? It ranges from one to nine. One being the least likely and nine being highly likely. Which option do you believe is not at all likely? -

0 1 2 3 4 5	6 7	8 9
Detractors	Neutral	Promoter

- NPS Net promoter score = % of Promoter - % of Detractors
- NPS will be in the range of -100 to 100

Difference of array and list

Array	List
Homogenous [Same type of data]	Heterogenous [Different type of data]
Fast to Generate and Extract Data	Slower Comparitively
Complex Calculation	Simple Calculation

Accessing elements of array using memory address

- a = a + i * memory
- a = 100 + 2 * 8 [We are going to access the index of 2 element]
- a = 116 [Memory location is 116 were we can find a]

When it comes to list it will get the address and using that address it will again search of element store

To check the time complexity Python has a magic function %timeit

1.0.1 Time taken to execute the arrays and list

[4]:
```
list = range(1000)
%timeit [i**2 for i in list]
```

```
194 µs ± 2.58 µs per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
```

[5]:
```
a = np.array(range(1000))
%timeit a**2
```

```
3.41 µs ± 131 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops each)
```

1.0.2 Creating an NumPy array

[12]:
```
a = np.array([[1,2,3,4,5],[3,5,6,4,7]])
```

1.0.3 Dimension of the Array

[13]:
```
a.ndim
```

[13]: 2

1.0.4 Shape of the Array

[14]:
```
a.shape
```

[14]: (2, 5)

1.0.5 Total values in the Array

[17]:
```
a.size
```

[17]: 10

[18]:
```
len(a)
```

[18]: 2

1.0.6 Masking

[19]:
```
b = np.array([1,2,3,4,5,6,7,8])
b[b>3]
```

[19]: array([4, 5, 6, 7, 8])

1.0.7 Exercise 1: Create a cubes in range 1000 with both array and list get the time difference

[20]:
```
#List
%timeit [x**3 for x in range(1000)]

#Array
c = np.array(1000)
%timeit c**3

d2 = np.array([[1,2,9,8],[5,6,7,4]])
print(d2.ndim)
print(d2.shape)
print(len(d2))
```

```
263 µs ± 16.6 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
2.37 µs ± 113 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops each)
2
(2, 4)
2
```

1.0.8 If we want to get the step size in float numpy comes handy

[26]:
```
[x for x in range(0,100,0.5)] # It will throw an error for list
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[26], line 1
----> 1 [x for x in range(0,100,0.5)] # It will throw an error for list

TypeError: 'float' object cannot be interpreted as an integer
```

[33]:
```
x = np.arange(0,100,0.5)
x
```

[33]:
```
array([ 0. ,  0.5,  1. ,  1.5,  2. ,  2.5,  3. ,  3.5,  4. ,  4.5,  5. ,
        5.5,  6. ,  6.5,  7. ,  7.5,  8. ,  8.5,  9. ,  9.5, 10. , 10.5,
       11. , 11.5, 12. , 12.5, 13. , 13.5, 14. , 14.5, 15. , 15.5, 16. ,
       16.5, 17. , 17.5, 18. , 18.5, 19. , 19.5, 20. , 20.5, 21. , 21.5,
       22. , 22.5, 23. , 23.5, 24. , 24.5, 25. , 25.5, 26. , 26.5, 27. ,
       27.5, 28. , 28.5, 29. , 29.5, 30. , 30.5, 31. , 31.5, 32. , 32.5,
       33. , 33.5, 34. , 34.5, 35. , 35.5, 36. , 36.5, 37. , 37.5, 38. ,
       38.5, 39. , 39.5, 40. , 40.5, 41. , 41.5, 42. , 42.5, 43. , 43.5,
       44. , 44.5, 45. , 45.5, 46. , 46.5, 47. , 47.5, 48. , 48.5, 49. ,
       49.5, 50. , 50.5, 51. , 51.5, 52. , 52.5, 53. , 53.5, 54. , 54.5,
       55. , 55.5, 56. , 56.5, 57. , 57.5, 58. , 58.5, 59. , 59.5, 60. ,
       60.5, 61. , 61.5, 62. , 62.5, 63. , 63.5, 64. , 64.5, 65. , 65.5,
```

```
66. , 66.5, 67. , 67.5, 68. , 68.5, 69. , 69.5, 70. , 70.5, 71. ,
71.5, 72. , 72.5, 73. , 73.5, 74. , 74.5, 75. , 75.5, 76. , 76.5,
77. , 77.5, 78. , 78.5, 79. , 79.5, 80. , 80.5, 81. , 81.5, 82. ,
82.5, 83. , 83.5, 84. , 84.5, 85. , 85.5, 86. , 86.5, 87. , 87.5,
88. , 88.5, 89. , 89.5, 90. , 90.5, 91. , 91.5, 92. , 92.5, 93. ,
93.5, 94. , 94.5, 95. , 95.5, 96. , 96.5, 97. , 97.5, 98. , 98.5,
99. , 99.5])
```

We have to use '&' when we are checking two different condition to single element, 'and' can be used for comparaing two elements

[48]:
```
x = np.arange(0,100,0.5)
x[(x%2 == 0) & (x%5 ==0)]
```

[48]:
```
array([ 0., 10., 20., 30., 40., 50., 60., 70., 80., 90.])
```