NumPy Array Attributes

In [1]:

```
import numpy as np
np.random.seed(0)  # seed for reproducibility

x1 = np.random.randint(10, size=6)  # One-dimensional array
x2 = np.random.randint(10, size=(3, 4))  # Two-dimensional array
x3 = np.random.randint(10, size=(3, 4, 5))  # Three-dimensional array
```

In [2]:

```
print("x3 ndim: ", x3.ndim)
print("x3 shape:", x3.shape)
print("x3 size: ", x3.size)
```

```
x3 ndim:  3
x3 shape: (3, 4, 5)
x3 size:  60
```

In [3]:

```
print("dtype:", x3.dtype)
```

```
dtype: int32
```

In [4]:

```
print("itemsize:", x3.itemsize, "bytes")
print("nbytes:", x3.nbytes, "bytes")
```

```
itemsize: 4 bytes
nbytes: 240 bytes
```

Array Indexing: Accessing Single Elements

In [5]:

```
x1
```

Out[5]:

```
array([5, 0, 3, 3, 7, 9])
```

In [6]:

```
x1[0]
```

Out[6]:

```
5
```

In [7]:

```
x1[4]
```

Out[7]:

```
7
```

In [8]:

```
x1[-1]
```

Out[8]:

```
9
```

In [9]:

```
x1[-2]
```

Out[9]:

```
7
```

In [10]:

```
x2
```

Out[10]:

```
array([[3, 5, 2, 4],
       [7, 6, 8, 8],
       [1, 6, 7, 7]])
```

In [11]:

```
x2[0, 0]
```

Out[11]:

```
3
```

In [12]:

```
x2[2, 0]
```

Out[12]:

```
1
```

In [13]:

```
x2[2, -1]
```

Out[13]:

```
7
```

In [14]:

```
x2[0, 0] = 12 # x2(0,0)の要素を変更する
x2
```

Out[14]:

```
array([[12,  5,  2,  4],
       [ 7,  6,  8,  8],
       [ 1,  6,  7,  7]])
```

In [15]:

```
x1[0] = 3.14159  # this will be truncated!
x1
```

Out[15]:

```
array([3, 0, 3, 3, 7, 9])
```

Array Slicing: Accessing Subarrays

x[start:stop:step]

In [17]:

```
x = np.arange(10)
x
```

Out[17]:

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [18]:

```
x[:5]  # first five elements
```

Out[18]:

```
array([0, 1, 2, 3, 4])
```

In [19]:

```
x[5:]  # elements after index 5
```

Out[19]:

```
array([5, 6, 7, 8, 9])
```

In [20]:

```
x[4:7]  # middle sub-array
```

Out[20]:

```
array([4, 5, 6])
```

In [21]:

```
x[::2]  # every other element
```

Out[21]:

```
array([0, 2, 4, 6, 8])
```

In [22]:

```
x[1::2]  # every other element, starting at index 1
```

Out[22]:

```
array([1, 3, 5, 7, 9])
```

In [23]:

```
x[::-1]  # all elements, reversed
```

Out[23]:

```
array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])
```

In [24]:

```
x[5::-2]  # reversed every other from index 5
```

Out[24]:

```
array([5, 3, 1])
```

In [25]:

```
### Multi-dimensional subarrays
x2
```

Out[25]:

```
array([[12,  5,  2,  4],
       [ 7,  6,  8,  8],
       [ 1,  6,  7,  7]])
```

In [26]:

```
x2[:2, :3]  # two rows, three columns
```

Out[26]:

```
array([[12,  5,  2],
       [ 7,  6,  8]])
```

In [27]:

```
x2[:3, ::2]  # all rows, every other column
```

Out[27]:

```
array([[12,  2],
       [ 7,  8],
       [ 1,  7]])
```

In [28]:

```
x2[::-1, ::-1]
```

Out[28]:

```
array([[ 7,  7,  6,  1],
       [ 8,  8,  6,  7],
       [ 4,  2,  5, 12]])
```

In [29]:

```
print(x2[:, 0])  # first column of x2
```

```
[12  7  1]
```

In [30]:

```
print(x2[0, :])  # first row of x2
```

```
[12  5  2  4]
```

In [31]:

```
print(x2[0])  # equivalent to x2[0, :]
```

```
[12  5  2  4]
```

In [32]:

```
#### Subarrays as no-copy views
```

In [33]:

```
print(x2)
```

```
[[12  5  2  4]
 [ 7  6  8  8]
 [ 1  6  7  7]]
```

In [34]:

```
x2_sub = x2[:2, :2]
print(x2_sub)
```

```
[[12  5]
 [ 7  6]]
```

In [35]:

```
x2_sub[0, 0] = 99
print(x2_sub)
```

```
[[99  5]
 [ 7  6]]
```

In [36]:

```
print(x2)
```

```
[[99  5  2  4]
 [ 7  6  8  8]
 [ 1  6  7  7]]
```

In [37]:

```
#### Creating copies of arrays
```

In [38]:

```
x2_sub_copy = x2[:2, :2].copy()
print(x2_sub_copy)
```

```
[[99  5]
 [ 7  6]]
```

In [39]:

```
x2_sub_copy[0, 0] = 42
print(x2_sub_copy)
```

```
[[42  5]
 [ 7  6]]
```

In [40]:

```
print(x2) #the original array is not touched
```

```
[[99  5  2  4]
 [ 7  6  8  8]
 [ 1  6  7  7]]
```

In [41]:

```
#### Reshaping of Arrays
```

In [42]:

```
grid = np.arange(1, 10).reshape((3, 3))
print(grid)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

In [43]:

```
x = np.array([1, 2, 3])

# row vector via reshape
x.reshape((1, 3))
```

Out[43]:

```
array([[1, 2, 3]])
```

In [44]:

```
# row vector via newaxis
x[np.newaxis, :]
```

Out[44]:

```
array([[1, 2, 3]])
```

In [45]:

```
# column vector via reshape
x.reshape((3, 1))
```

Out[45]:

```
array([[1],
       [2],
       [3]])
```

In [46]:

```
# column vector via newaxis
x[:, np.newaxis]
```

Out[46]:

```
array([[1],
       [2],
       [3]])
```

In [47]:

```
#### Array Concatenation and Splitting
```

In [48]:

```
#Concatenation of arrays
```

In [49]:

```
x = np.array([1, 2, 3])
y = np.array([3, 2, 1])
np.concatenate([x, y])
```

Out[49]:

```
array([1, 2, 3, 3, 2, 1])
```

In [50]:

```
z = [99, 99, 99]
print(np.concatenate([x, y, z]))
```

```
[ 1  2  3  3  2  1 99 99 99]
```

In [51]:

```
grid = np.array([[1, 2, 3],
                 [4, 5, 6]])
```

In [52]:

```
# concatenate along the first axis
np.concatenate([grid, grid])
```

Out[52]:

```
array([[1, 2, 3],
       [4, 5, 6],
       [1, 2, 3],
       [4, 5, 6]])
```

In [53]:

```
# concatenate along the second axis (zero-indexed)
np.concatenate([grid, grid], axis=1)
```

Out[53]:

```
array([[1, 2, 3, 1, 2, 3],
       [4, 5, 6, 4, 5, 6]])
```

In [54]:

```
x = np.array([1, 2, 3])
grid = np.array([[9, 8, 7],
                 [6, 5, 4]])

# vertically stack the arrays
np.vstack([x, grid])
```

Out[54]:

```
array([[1, 2, 3],
       [9, 8, 7],
       [6, 5, 4]])
```

In [55]:

```
# horizontally stack the arrays
y = np.array([[99],
              [99]])
np.hstack([grid, y])
```

Out[55]:

```
array([[ 9,  8,  7, 99],
       [ 6,  5,  4, 99]])
```

In [56]:

```
#Splitting of arrays
```

In [57]:

```
x = [1, 2, 3, 99, 99, 3, 2, 1]
x1, x2, x3 = np.split(x, [3, 5])
print(x1, x2, x3)
```

```
[1 2 3] [99 99] [3 2 1]
```

In [58]:

```
grid = np.arange(16).reshape((4, 4))
grid
```

Out[58]:

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

In [59]:

```
upper, lower = np.vsplit(grid, [2])
print(upper)
print(lower)
```

```
[[0 1 2 3]
 [4 5 6 7]]
[[ 8  9 10 11]
 [12 13 14 15]]
```

In [60]:

```
left, right = np.hsplit(grid, [2])
print(left)
print(right)
```

```
[[ 0  1]
 [ 4  5]
 [ 8  9]
 [12 13]]
[[ 2  3]
 [ 6  7]
 [10 11]
 [14 15]]
```

In [61]:

```
#A "comb" function
N, n = 101, 5
def f(i):
    return (i % n == 0) * 1
comb = np.fromfunction(f, (N,), dtype=int)
print(comb)
```

```
[1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0
 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0
 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1]
```

In [62]:

```
```

Creating a Magic Square

魔方陣は、各行、列、主対角線のエントリが同じ数になるように合計したN×Nのグリッドのグリッドです（N（N^2 + 1）/ 2と等しい）。 奇数Nの魔方陣を構築する方法

以下のとおりであります：

- ステップ1.上の行の中央から始め、n = 1とします。
- ステップ2. nを挿入する: 現在のグリッド位置に;
- ステップ3. n = N^2の場合: グリッドが完成したので停止します。 それ以外の場合はnをインクリメントします;
- ステップ4.グリッドの外側に移動する場合は、最初の列または最後の行に折り返して、対角線上および右に移動します。 このセルがすでに入力されている場合は、代わりに1スペース下に垂直に移動します。
- ステップ5.ステップ2に戻ります。

In [63]:

```
# Create an N x N magic square. N must be odd.
import numpy as np

N  = 3
magic_square = np.zeros((N,N), dtype=int)

n = 1
i, j = 0, N//2

while n <= N**2:
    magic_square[i, j] = n
    n += 1
    newi, newj = (i-1) % N, (j+1)% N
    if magic_square[newi, newj]:
        i += 1
    else:
        i, j = newi, newj

print(magic_square)
```

```
[[8 1 6]
 [3 5 7]
 [4 9 2]]
```

数独グリッドの有効性を確認する

In [64]:

```
import numpy as np

def check_sudoku(grid):
    """ Return True if grid is a valid Sudoku square, otherwise False. """
    for i in range(9):
        # j, k index top left hand corner of each 3x3 tile
        j, k = (i // 3) * 3, (i % 3) * 3
        if len(set(grid[i,:])) != 9 or len(set(grid[:,i])) != 9¥
                or len(set(grid[j:j+3, k:k+3].ravel())) != 9:
            return False
    return True

sudoku = """145327698
            839654127
            672918543
            496185372
            218473956
            753296481
            367542819
            984761235
            521839764"""
# Turn the provided string, sudoku, into an integer array
grid = np.array([[int(i) for i in line] for line in sudoku.split()])
print(grid)

if check_sudoku(grid):
    print('grid valid')
else:
    print('grid invalid')
```

```
[[1 4 5 3 2 7 6 9 8]
 [8 3 9 6 5 4 1 2 7]
 [6 7 2 9 1 8 5 4 3]
 [4 9 6 1 8 5 3 7 2]
 [2 1 8 4 7 3 9 5 6]
 [7 5 3 2 9 6 4 8 1]
 [3 6 7 5 4 2 8 1 9]
 [9 8 4 7 6 1 2 3 5]
 [5 2 1 8 3 9 7 6 4]]
grid valid
```

In [65]:

```
## その他
```

In [ ]: