Data structures in Python

- Numbers
- Strings
- Lists
- Tuples
- Dictionary
- Structured arrays in Numpy
- Structured arrays in Pandas

Numbers

- Store numeric values
- Immutable data type
  - Cannot change the value, so changing the value of a number data results in a newly allocated object
- Support four numerical types
  - int, long, float, complex
  - E.g., 10 (int), 51924232L (long), 0.2 (float), 3.14j (complex)
  - Type conversion: int(x), long(x), float(x), complex(x)

```
var1 = 1    var2 = float(var1)
```

Strings

- Data type for a string
- Can create a string by enclosing characters in quotes or double quotes

```
var1 = 'Hello World!'
var2 = "Python Programming“
var1[0]: H
var2[1:3]: yt
var2[3:4]: h
var1[:3]: Hel
```

Lists

- Sequence or array data
- An element can be accessed by an index
  - Index begins with zero
- Written as a list of common-separated values between square brackets
- Items are not necessary to be of the same type in a list

```
list1 = ['computer', 2000];
list2 = [1, 2, 3, 4, 5 ];
```

Lists

- Accessing values in Lists
  - With index or indices

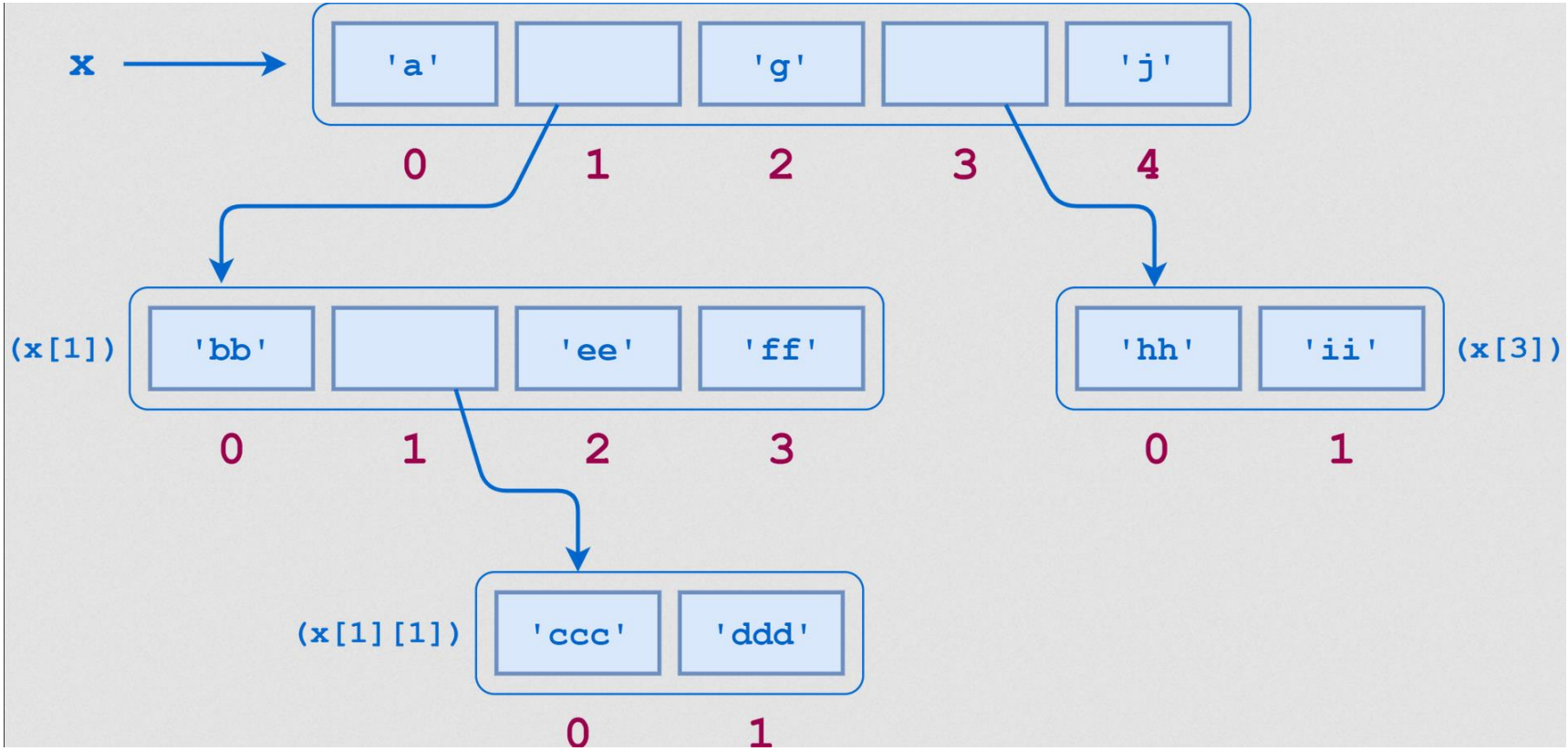
```
list2 = [1, 2, 3, 4, 5 ];
list2[0]: 1
list2[1:5]: [2, 3, 4, 5]
list2[-1]: 5
list2[-2:-1]: [4]
list2[:3]: [1, 2, 3]
list2[2:]: [3, 4, 5]
```

- Updating values

```
list2[2:4] = [1, 2]
list2[:] = [1, 2, 1, 2, 5]
```

Nested List

x = ['a', ['bb', ['ccc', 'ddd'], 'ee', 'ff'], 'g', ['hh', 'ii'], 'j']



Ref: https://realpython.com/python-lists-tuples/

Tuples

- Collection of objects; ordered and immutable
- Comma-separated values between parentheses

```
tup1 = ('physics', 'chemistry', 2000);
tup2 = (1, 2, 3, 4, 5, 6, 7 );
```

- Efficient for memory and debugging
- Inefficient for appending data

Dictionary

- Each item consists of a key and a value

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
dict['Name']: 'Zara'
dict['Age']: 7
```

NumPy's multidimensional array

- Homogeneous multidimensional array
  - All elements of the same type

```
>>> import numpy as np
>>> a = np.arange(15).reshape(3, 5)
>>> a
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
>>> a.shape
(3, 5)
>>> a.ndim
2
```

```
>>> a.dtype.name
'int64'
>>> a.itemsize
8
>>> a.size
15
>>> type(a)
<class 'numpy.ndarray'>
```

Ref: https://numpy.org/doc/stable/user/quickstart.html

NumPy's multidimensional array

- Creating an array

```
import numpy as np
a = np.array([2,3,4])
b = np.array([(1.5,2,3), (4,5,6)])
c = np.array( [ [1,2], [3,4] ], dtype=complex)
np.zeros((3, 4))
np.ones( (2,3,4), dtype=np.int16 )
np.arange( 10, 30, 5 ) -> array([10, 15, 20, 25])
```

Ref: https://numpy.org/doc/stable/user/quickstart.html

NumPy's multidimensional array

```
>>> a = np.arange(6)                         # 1d array
>>> print(a)
[0 1 2 3 4 5]
>>>
>>> b = np.arange(12).reshape(4,3)           # 2d array
>>> print(b)
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
```

NumPy's multidimensional array

```
>>> c = np.arange(24).reshape(2,3,4)          # 3d array
>>> print(c)
[[[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]]

 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]]
```

NumPy's multidimensional array

```
>>> A = np.array( [[1,1],
...                [0,1]] )
>>> B = np.array( [[2,0],
...                [3,4]] )
>>> A * B
array([[2, 0],
       [0, 4]])
>>> A @ B
array([[5, 4],
       [3, 4]])
>>> A.dot(B)
array([[5, 4],
       [3, 4]])
```

$$A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, B = \begin{bmatrix} 2 & 0 \\ 3 & 4 \end{bmatrix}$$

$$A * B = \begin{bmatrix} 1*2 & 1*0 \\ 0*3 & 1*4 \end{bmatrix}$$

$$A \,@\, B = A.\mathrm{dot}(B) = \begin{bmatrix} 1*2+1*3 & 1*0+1*4 \\ 0*2+1*3 & 0*0+1*4 \end{bmatrix}$$

NumPy's multidimensional array

- Universal Functions
  - functions operate element wise on an array

```
>>> B = np.arange(3)
>>> B
array([0, 1, 2])
>>> np.exp(B)
array([1.        , 2.71828183, 7.3890561 ])
>>> np.sqrt(B)
array([0.        , 1.        , 1.41421356])
>>> C = np.array([2., -1., 4.])
>>> np.add(B, C)
array([2., 0., 6.])
```

NumPy's multidimensional array

- Indexing with Boolean Arrays

```
>>> a = np.arange(12).reshape(3,4)
>>> b = a > 4
>>> b
s shape
array([[False, False, False, False],
       [False,  True,  True,  True],
       [ True,  True,  True,  True]])
>>> a[b]
cted elements
array([ 5,  6,  7,  8,  9, 10, 11])
```

Pandas

- Series
  - One-dimensional labeled array of any data type

```
import numpy as np
import pandas as pd
s = pd.Series([1, 3, 5, np.nan, 6, 8])
s = pd.Series(np.random.randn(5),
        index=['a', 'b', 'c', 'd', 'e'])
d = pd.Series({'b': 1, 'a': 0, 'c': 2})
```

Pandas' DataFrame

- 2-dimensional labeled data structure with columns of potentially different types

```
d = {'one': pd.Series([1., 2., 3.], index=['a', 'b', 'c']),
     'two': pd.Series([1., 2., 3., 4.], index=['a', 'b', 'c', 'd'])}
df = pd.DataFrame(d)
```

```
  one  two
a  1.0  1.0
b  2.0  2.0
c  3.0  3.0
d  NaN  4.0
```

Pandas' DataFrame

```
data = np.zeros((2, ), dtype=[('A', 'i4'), ('B', 'f4'), ('C', 'a10')])
data[:] = [(1, 2., 'Hello'), (2, 3., "World")]
pd.DataFrame(data)
```

	A	B	C
0	1	2.0	b'Hello'
1	2	3.0	b'World'