

Learning Python

Getting results for beamlines and scientific programming

2. Basic Python: higher level data types

Outline of topics to be covered

1. High-level data types
 - a) Lists
 - b) Tuples
 - c) Strings revisited
 - d) Dicts (dictionaries)
 - e) Sets
2. Length of a collection (len)



Higher-level data types in Python

Python offers a number of data types that provide ways to form collections of other data types. Each of these have advantages for different types of programming tasks.

The three most basic higher-level data types in Python are **lists**, **tuples** and **dicts**.

- There are many other types of higher-level data types in Python that can be very useful, such as sets and datetime

Basics:

- lists and tuples are ordered collections of values,
- dictionaries (dicts) are unordered arrays of values with user supplied keys
- Lists, tuples, dicts can contain as values primitives (int, floats, etc.) or any higher-level data type (lists inside dicts inside tuples...)
- Lists, tuples and dictionaries all can play the role of arrays in Fortran or C.

Lets look in more detail, starting with the list data type



Higher-level data types: Lists

A list contains a bunch of stuff in a specified order (Downey, Chapter 8)

- List:
a = [1,2,3,5,7,11]
- Elements in a list can be addressed individually:
print a[2]
3 (Note: indexing starts with [0])
- List elements can be changed:
a[2] = -3
print a
[1, 2, -3, 5, 7, 11]
- A list can contain any other Python data type
cvariable = complex(0,1)
B = [1, 3.14159, 'pi', ['a','list','in','a','list'], cvariable]



Slicing a list

The indices on a list can address a large number parts of the array in many ways using [start:end:increment]

```
>>> a = [1,2,3,5,7,11]
>>> a
[1, 2, 3, 5, 7, 11]
>>> a[1:4]
[2, 3, 5]
>>> a[4:]
[7, 11]
>>> a[:4]
[1, 2, 3, 5]
>>> a[2:-1]
[3, 5, 7]
```

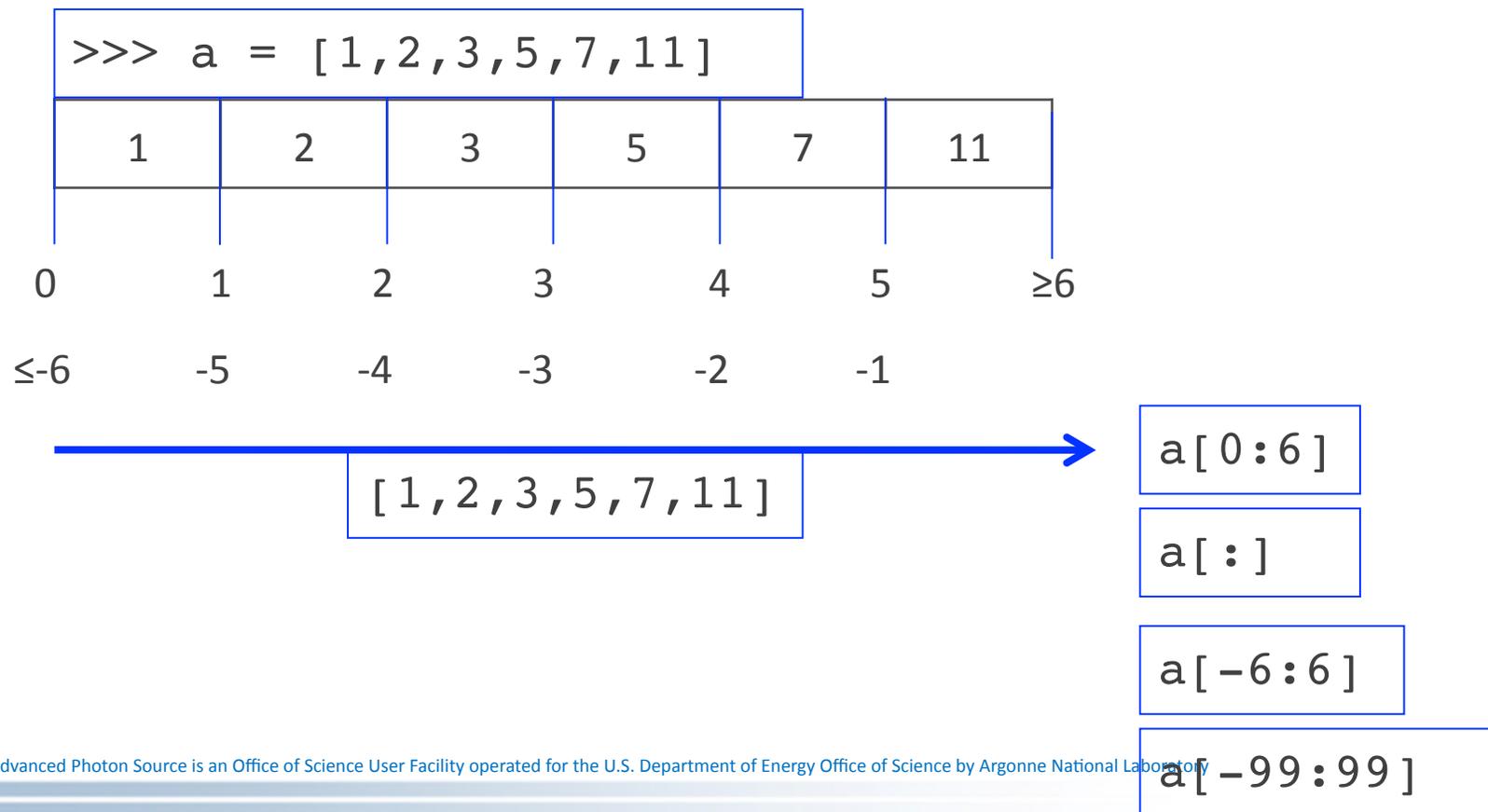
```
>>> a[2:-1]=[0,0,0]
>>> print a
[1, 2, 0, 0, 0, 11]
>>> a[1::2]
[2, 0, 11]
>>> a[::2]=['a','b','c']
>>> a
['a', 2, 'b', 0, 'c', 11]
>>> a[:]
['a', 2, 'b', 0, 'c', 11]
```

```
>>> a = [1,2,3,5,7,11,]
>>> a[-3:-1]
[5, 7]
>>> a[-1:-3]
[]
>>> a[::-1]
[11, 7, 5, 3, 2, 1]
>>> a[-1:-3:-1]
[11, 7]
```



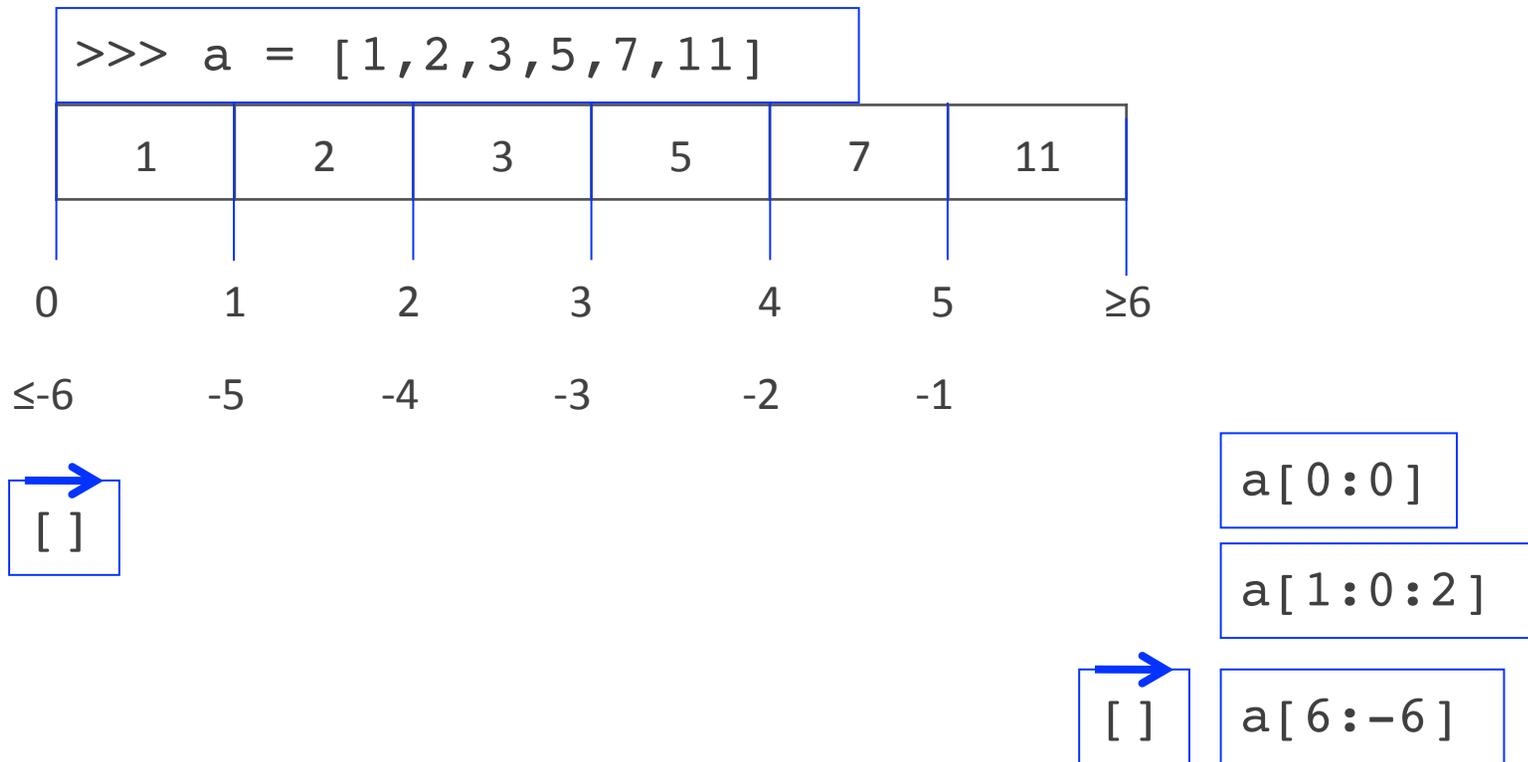
Understanding indexing

Indexing can be rationalized as referring to nodes, which can be numbered as shown on the diagram below. Note that most nodes can be reference from the collection start (0 or positive) or from the end (negative). It is fine to use an index that is out of range.



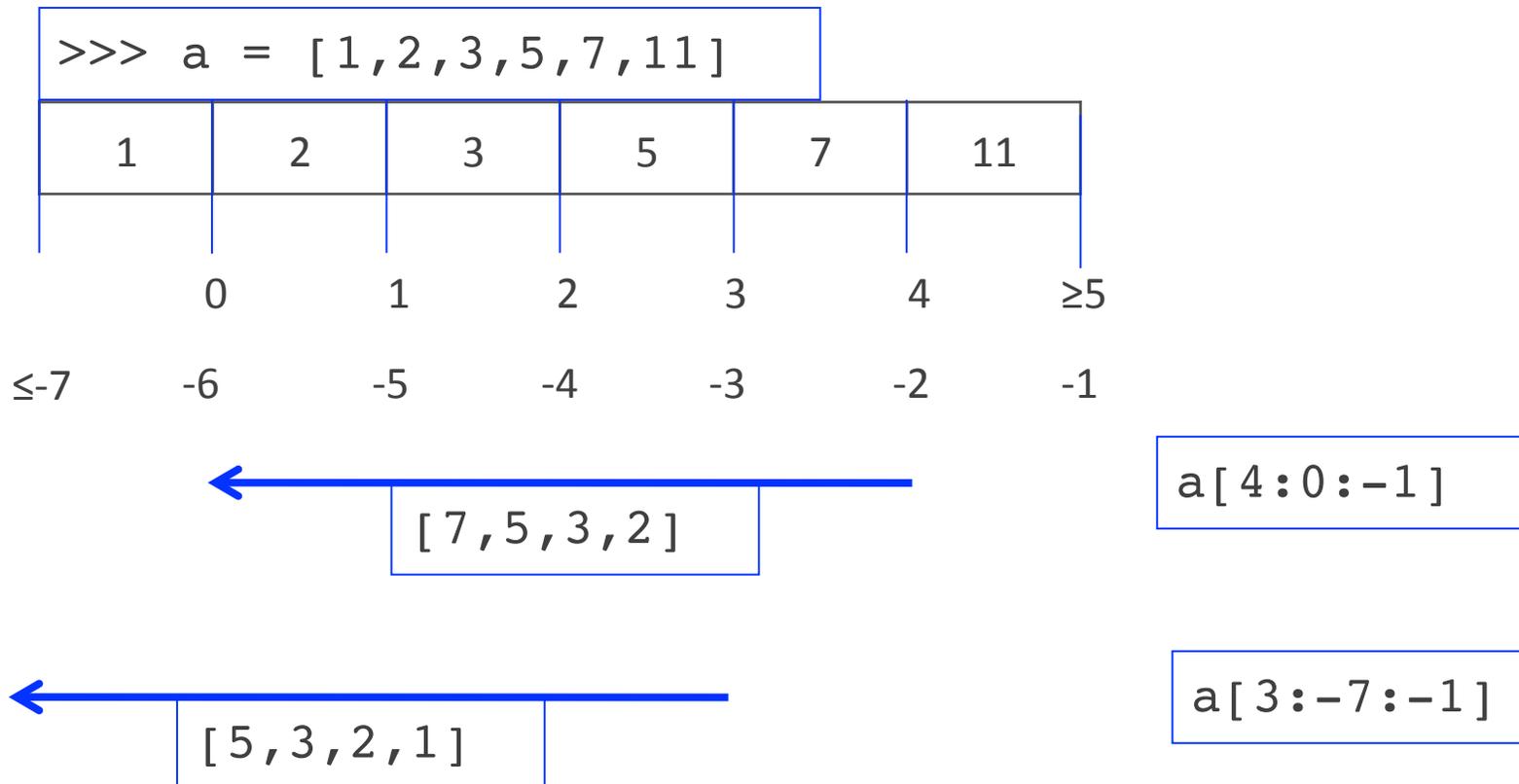
Start and end index are directional

Indexing goes from “left” to “right” with positive increment (default). If the start is after the end, the result is an empty collection.



Use of a negative increment changes this

If a negative index is used, the node addresses shift and the indexing goes from “right” to “left”.



More fun things with lists

All Python data types are actually objects, with associated “methods”.

To see a list, use `print dir(var)`; ignore items beginning with “_”

Method for lists: `append`, `count`, `extend`, `index`, `insert`, `pop`, `remove`, `reverse`, `sort`

```
>>> a
[1, 2, 3, 5, 7, 11]
>>> a.append(13)
>>> a
[1, 2, 3, 5, 7, 11, 13]
>>> a.insert(0,17)
>>> a
[17, 1, 2, 3, 5, 7, 11, 13]
>>> a.sort()
>>> a
[1, 2, 3, 5, 7, 11, 13, 17]
```

```
>>> a = [1, 2, 3, 5, 7, 11, 13, 17]
>>> a.pop()
17
>>> a
[1, 2, 3, 5, 7, 11, 13]
>>> a.pop(0)
1
>>> a
[2, 3, 5, 7, 11, 13]
>>> len(a)
6
```



Building a list

- A common way to use lists is to start with an empty list and add elements

```
A = []  
A.append(11)  
A.append(13)
```

- It is not possible to add elements by addressing past the end of the list:

```
A[2] = 17  
IndexError: list assignment index out of range
```

- Like strings, lists can be added:

```
>>> [1,2,3] + [4,5,6]  
[1, 2, 3, 4, 5, 6]
```



Higher-level data types: Tuples

Tuples (Downey, Chapter 9) are very similar to lists, but tuples cannot be changed (immutable)

- Tuples:
 - Note that any time items are joined with a comma, a tuple is created, but this tends to be confusing; it is wise to always use parenthesis:

```
t = 1,2,3,5,7,11 # or
```

```
t = (1,2,3,5,7,11,) # or
```

```
t = (1,2,3,5,7,11)
```

- Elements in a tuple can be addressed individually just like a list:

```
print t[2]
```

```
3
```

- Note: indexing starts with [0], same as with a list

- Important distinction: Tuple elements **cannot** be changed:

```
>>> t[2] = -3
```

```
TypeError: 'tuple' object does not support item assignment
```



Higher-level data types: Tuples

- A tuple can contain any other Python data type
cvariable = complex(0,1)
B = (1, 3.14159, 'pi', ['a','list','in','a','tuple'], cvariable)
- One can define an empty tuple: t = ()
 - One can't add to it
- To define a tuple with one element, the “extra” comma is required
t = (1,)
- Tuples can be added as well

```
>>> (1,2,3) + (4,5,6)
(1, 2, 3, 4, 5, 6)
```

- Note that adding two tuples creates a third distinct tuple but does not change the original ones



Quick Review: Tuples vs. Lists

- Tuples optionally use parenthesis () and cannot be changed
- Lists always use square brackets [] and can be modified in place
- They are different data types. You can't add a tuple and a list:

```
>>> (1,2,3) + [4,5,6]  
TypeError: can only concatenate tuple (not "list") to tuple
```



Reprise: Strings

- Strings can be indexed just like lists and tuples
- Strings cannot be changed -- like tuples (but unlike lists)

```
>>> abc = 'abcdefghijklmnopqrstuvwxy'
>>> len(abc)
26
>>> abc[1]
'b'
>>> abc[::-2]
'zxvtrpnljhfdb'
>>> abc[1] = 'B'
TypeError: 'str' object does not support item assignment
>>>
```



Python Dictionaries

- A Python dictionary (type dict) is a place where any number of data items can be stored, associated with a key (think of the key as a label that identifies each storage location). Unlike a list or tuple, where items are stored and retrieved sequentially, there is no order associated with elements in a dict.
- For some uses a dict is very much like an array
- Dicts can be defined by adding elements or all at one time (or both). The two examples below define the same dictionary:

```
A = {}  
A[1] = 1.0  
A[2] = 2.0  
A['two'] = 2  
A[2.1] = 2
```

```
A = { 1:1.0,  
      2: 2.0,  
      'two': 2,  
      2.1: 2,  
      }
```

```
>>> A  
{1: 1.0, 2: 2.0, 2.1000000000000001: 2, 'two': 2}  
>>> A[2.1]  
2  
>>> A[2]  
2.0
```



Methods for dicts

- The routines associated with a dict are: clear, copy, fromkeys, get, has_key, items, iteritems, iterkeys, itervalues, keys, pop, popitem, setdefault, update, values

Most commonly used:

keys() gives a list of keys

values() gives a list of values in the same order as keys()

items() gives the dict contents as key, value pairs

has_key(key) is True if the key is defined

get(key) returns the value or None

get(key,d) returns the value or d

```
>>> A = {1:1.0, 2: 2.0, 'two': 2, 2.1: 2}
>>> A.keys()
[1, 2, 2.1000000000000001, 'two']
>>> A.values()
[1.0, 2.0, 2, 2]
>>> A.items()
[(1, 1.0), (2, 2.0), ...
>>> A.has_key(2)
True
>>> A.has_key(2.2)
False
>>> A.get(2, 'not specified')
2.0
>>> A.get(2.2, 'not specified')
'not specified'
```



A newer Python data type: sets

- A set is similar to a list, except that it is unordered and only the unique elements are saved:

```
>>> set([3,2,1,0,1,2,3])
set([0, 1, 2, 3])
```

- Working with sets is a very nice way to perform logical manipulations of groups of items

```
>>> s1 = set([3,2,1,0,1,2,3])
>>> s2 = set((3,4,5))
>>> s2
set([3, 4, 5])
>>> s1 | s2 # union of sets
set([0, 1, 2, 3, 4, 5])
>>> len(s1 & s2) # count shared elements
1
```

- Conversion back to a list is easy

```
>>> list( s1|s2 ) # union of sets as a list
[0, 1, 2, 3, 4, 5]
```



The len() built-in

- Use len(var) to find out the number of elements in a list, tuple or dict or characters in a string

```
>>> A = []
>>> len(A)
0
>>> A = (1,2,3)
>>> len(A)
3
>>> A = {1:1.0, 2: 2.0, 'two':
        2, 2.1: 2}
>>> len(A)
4
>>> len('test')
4
```



Homework

- Specify the data types of the following values:

`(1,2,3)`

`[2,3,4]`

`'To be, or not to be'`

`'To be', 'or not to be'`

`('To be', 'or not to be')`

`{2:3, 3:4}`

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

