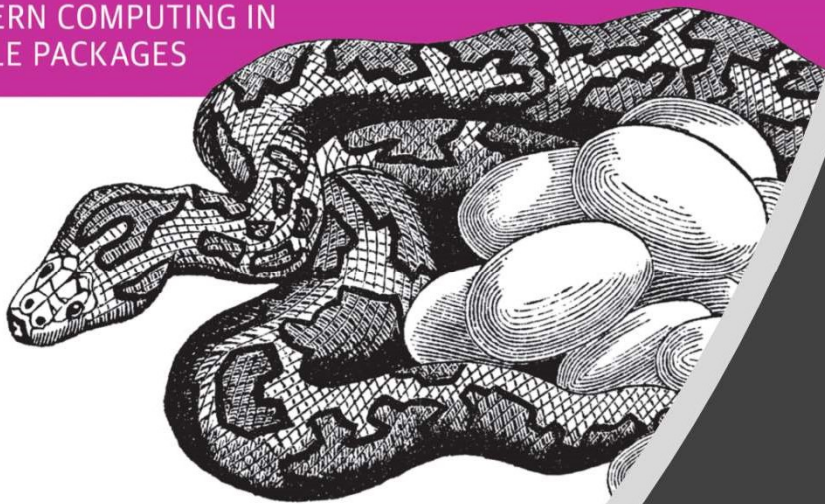


Introducing Python

MODERN COMPUTING IN
SIMPLE PACKAGES



Week 3.

Python for Data Science:

Sequence Types and Dictionaries

Week 3: Sequence Types & Dictionaries

1. Checking In
2. Sequences
3. Lists
4. Ranges, Tuples, and Sets (activity 1)
5. Dictionaries (activity 2)
6. Mutability and Pitfalls (activity 3)

We mayn't complete the breakouts in class - that's okay - keep working on the challenges and feel free to send your solutions around the Google Group.

1. Checking In

- How is it going?
- What was the hardest part of the homework? The easiest?

2. Sequences

What are they?

p y t h o n

What's *not* a sequence?

2. Sequences

What are they?

python

In Python, a sequence is a generic term for an ordered group of objects.

Examples include

- lists,
- tuples, *and*
- strings.

What's *not* a sequence?

Any data types without an inherent order, such as

- dictionaries,
- sets,
- ints, *and*
- floats.

Sequences

<https://docs.python.org/3/tutorial/datastructures.html>
<https://docs.python.org/3/library/stdtypes.html>

- Share a lot of common methods (but not all): include or slice with []. The offset starts with 0.

E.g., `myList = ['Lars', 'Juan', 'Pierre', 'Marie', 'Tan']`

- How many elements in the list?

`len(myList)`

- I'm looking for "Lars" - is he in the list?

`"'Lars' in myList"`

```
>>> mylist = ['lars', 'fish']  
>>> print(('lars' in mylist))  
True
```

- How to add (concatenate) 2 lists?

```
a = list()  
a.append("tom")  
a[0] = "tom"
```

Sequences

- Sequences are helpful! Their methods tell us about the size, min/max values, counts (occurrences) of an object, and more!
- `min()` `max()` `myList['a', 'b', 'c', 'd']`
- `myList.index('x')` # locate the first instance of “x”
- `myList.count('x')` # how many times of “x”

What's up with the dot? And the parentheses?

Sequences

- Sequences are helpful! Their methods tell us about the size, min/max values, counts (occurrences) of an object, and more!
- `min()` `max()` `myList['a', 'b', 'c', 'd']`
- `myList.index('x')` # locate the first instance of “x”
- `myList.count('x')` # how many times of “x”

What's up with the dot? And the parentheses?

Parentheses hold the arguments we're passing to the function, e.g., `len(myList)`. Not all functions require arguments.

The dot notation includes that a function is defined in the specific object before the dot.

Sequences: lists, ranges, tuples & sets

- Lists are really helpful and iterating thru lists can change how we program because sequences may return `True/False` but equally may return a value or content.
- Mutability: a list is mutable, meaning we can change the length and content. E.g.,
 - `alist = []`
 - `alist.append('cat')`
 - `blist = ['a', 'b', 'c']`
 - `alist.extend(blist)`
 - `print(alist)`
`['cat', 'a', 'b', 'c']`
- What data types are mutable? Immutable?

Sequences: lists, ranges, tuples & sets

- Lists are really helpful and iterating thru lists can change how we program because sequences may return `True/False` but equally may return a value or content.
- Mutability: a list is mutable, meaning we can change the length and content. E.g.,
 - `alist = []`
 - `alist.append('cat')`
 - `blist = ['a', 'b', 'c']`
 - `alist.extend(blist)`
 - `print(alist)`
`['cat', 'a', 'b', 'c']`
- What data types are mutable? Immutable?

Mutability means the object can be changed; dictionaries, lists, and sets are mutable; tuples and strings are not. Primitive data types such as `int` and `float` are also immutable.

Composite Types

- A list is a composite type - what does that mean?
- Other examples?
 - A composite type is comprised of other types. Lists, tuples, dictionaries are composite because they can hold or contain other objects. int, floats, strings are not composites.

• E.g., `demo = ['cat', 'dog', 33, ['j', 'k', 'l']]`



• `print(demo[3])` \rightarrow `'j', 'k', 'l'`

- `myList.insert(index, value)`
- `myList.pop(x)` *# pops last value by default but can take instead index argument "x"*
- `myList.remove()` *# use remove to eliminate the first instance of a value*
- `myList.sort()` *# mutate the list - sorting by default in ascending order*
- `sorted(myList)` *# this returns a new list*
- `myList.reverse()` *# reverses the list.*

```
myList.append("x")           # adds "x" to the end of the list
```

```
myList.extend(otherList)     # adds items from otherList to  
the end of myList
```

```
myList[a] = "z"              #swaps out the item at index [a] with  
whatever z stands for
```

```
myList.clear()               #clears out the list elements
```

```
del (myList[x])              #  deletes item from index x
```

RANGES

a sequence

need to be listed to yield the elements

`range(start, stop, step)`

TUPLES

a sequence

like a list but immutable

instantiate: `tup_X=(1, 2, 3)` or

`tup(1, 2, 3)`

Can use a tuple to create multiple objects

SETS

Unordered and mutable

*Unique, keys only

```
>>> a=range(0,9)
>>> a
range(0, 9)
>>> list(a)
[0, 1, 2, 3, 4, 5, 6, 7, 8]
>>> type(a)
<class 'range'>
>>> type(list(a))
<class 'list'>
```

```
>>> low, high = 10,20
>>> print(low, high)
10 20
>>>
```

btw ... tuples

- Tuples are like lists - but they are immutable. What's happening here?

```
>>> a=([1,2,3],2,3)
>>> type(a)
<class 'tuple'>
>>> a[0].append(5)
>>> a
([1, 2, 3, 5], 2, 3)
>>> type(a)
<class 'tuple'>
>>> a[1]=10
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Breakout Rooms (1)

`range(start, stop(exclusive), step)` - Practice creating these outputs.

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

`[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`

`[2, 4, 6, 8, 10, 12]`

`[2, 4, 6, 8, 10, 12, 13, 14, 15, 17, 19, 21]`

`[-1, 0, 1, 2, 3]`

`[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]`

Dictionaries

Mutable, what does that imply?

Not a sequence, what does that mean?

Maps keys to values

aka: map, key:value store

```
a = {'fred':1, 'frank':3, 'ben':1}    book = {'changjing':'555-1212',  
'jim':'333-234'}
```

```
a = {'names': {'fred':404, 'frank':3, 'ben':1}} # can be nested (JSON)
```

Values can be any type

Keys need to be hashable

should be immutable

Python uses a hash function to quickly locate items stored in a dictionary. The key, when passed through the hash function, points to a unique place in the computer's memory. This makes finding the value extremely fast. Keys cannot be mutable, since if they were, the hash function would not return the same result.

Dictionaries

- Instantiation (many ways)

```
dict_x = dict('fred'=1, 'frank'=3, 'ben'=1)
```

```
dict_x = {'fred':1, 'frank':3, 'ben':1}
```

Above is a dict literal

```
dict_x = dict([('fred':1),('frank',3),('ben',1)])
```

As a list of tuples

- Index by key to get value

```
dict_x['fred']
```

indexing by key name

```
dict_x.fred
```

dot notation when there are no spaces

Dictionaries | More Methods

```
del(dict_x['key'])
```

delete by key reference

```
dict_x.pop('key', 'default value')
```

pop the value for key from dictionary. If the key doesn't exist, the function will return the default

```
dict_x.get('key', 'default value')
```

```
dict_x.clear()
```

```
dict_x.update(dict2)
```

appends a second diction to the first

```
dict_x.keys()
```

```
dict_x.values()
```

get the key:value pairs

```
dict_x.items()
```

Sample .json structures

```
{ people
  [name, tom]
  [name, fifi]
}
```

```
{ "menu":
  { "id": "file",
    "value": "File",
    "popup":
      { "menuitem": [
          { "value": "New",
            "onclick": "CreateNewDoc()" }
        ],
        { "value": "Open",
          "onclick": "OpenDoc()" }
        ],
        { "value": "Close",
          "onclick": "CloseDoc()" }
        ]
      }
  }
}
```

<https://json.org/example.html>

List & Dictionary Activity

We are now going to solve a very popular problem: How do you count the words in a document?

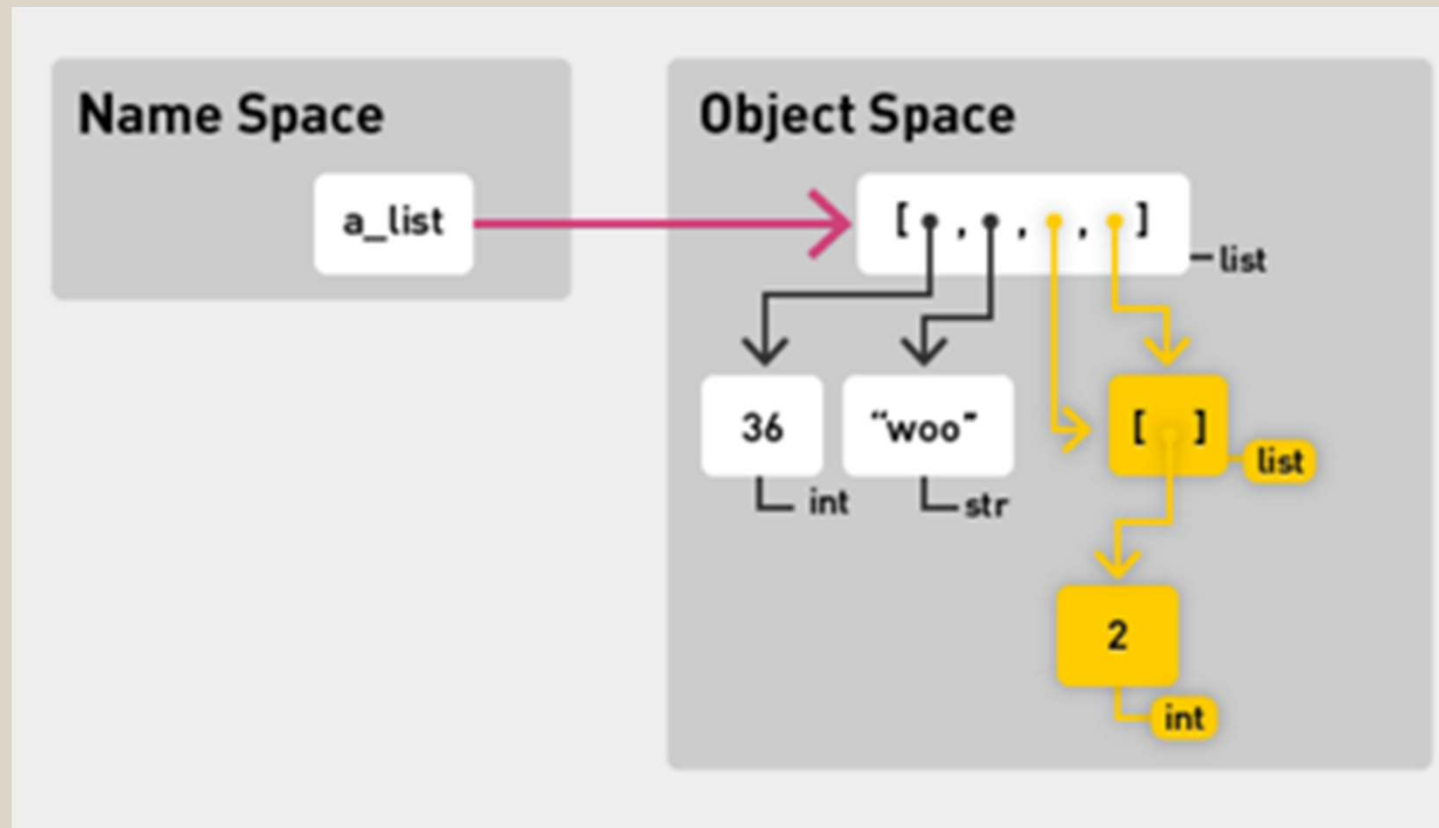
While the solution here is simple, you will see in later courses that this is an excellent first problem when learning how to massively parallelize your code across a cluster of computers.

The activity will guide you to the solution in a series of steps.

As you will see next week, the “while” loop in this activity could be better represented by a “for” loop. For now, please work with the “while” loop.

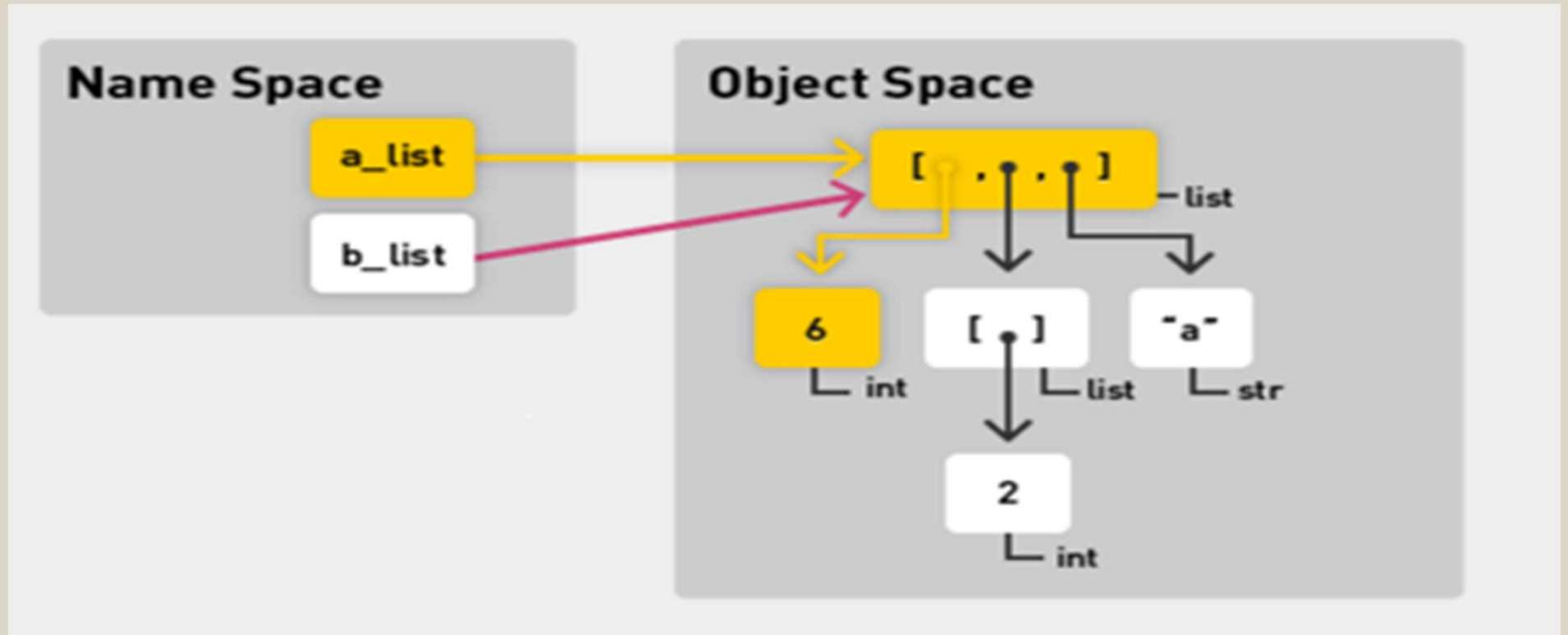
Mutability gotchas ...

1: This list points to the same object

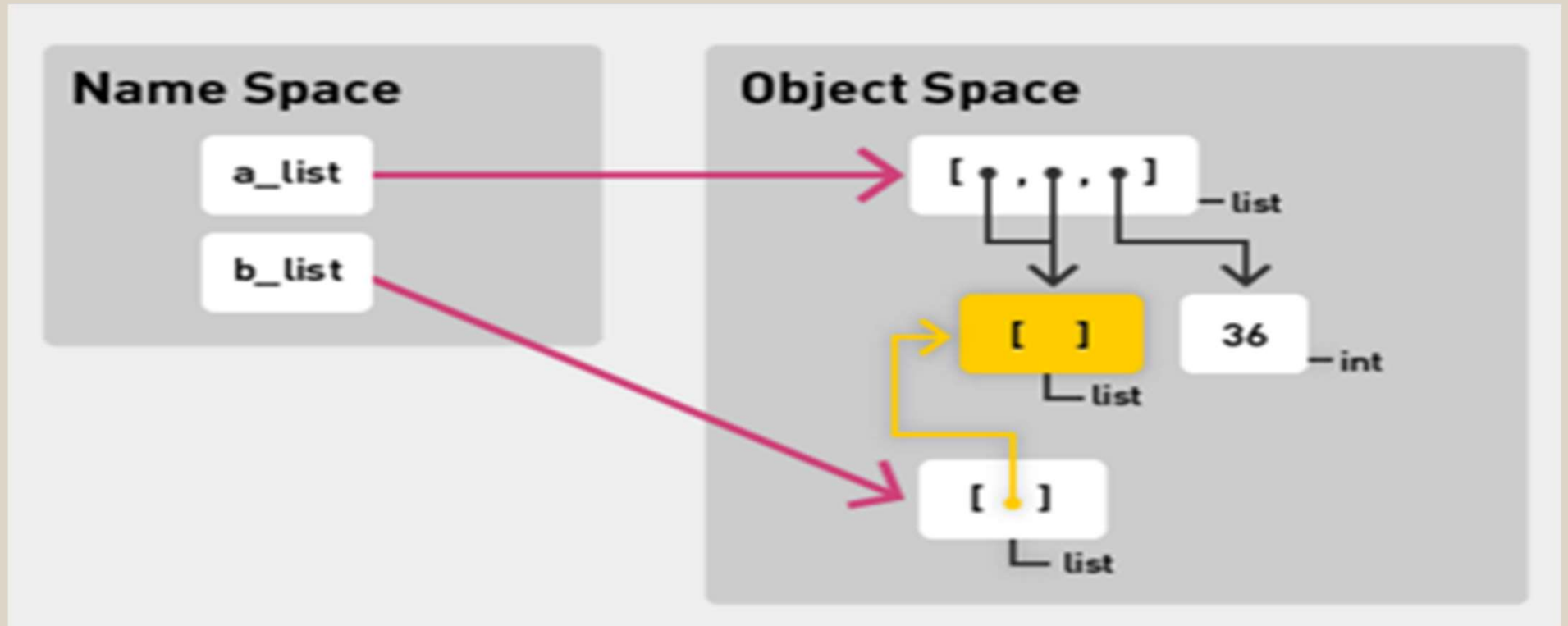


that is, items 3 and 4 are *the same object*: `[36, "woo", [2], [2]]`

Mutability Gotchas (2)



Mutability Gotchas (3)



Copy and Deepcopy

- Consider this snippet:

```
ls_x = [ 1, 2, 3, [ 'Frank', 'Fred' ] ]  
ls_x_cp = ls_x.copy()  
from copy import deepcopy  
ls_x_deep = deepcopy(ls_x)  
ls_x[3][1] = 'Mufasa'
```

Copy & Deepcopy

- What is copy?
 - Copy creates an independent copy of all list elements at the first level of the list.
- How does copy differ from deepcopy?
 - *Deepcopy creates an independent copy of all list elements at all levels.*
- What is the final value of `Ls_x_cp` and `Ls_x_deep`?
 - `Lx_x_cp` is `[1,2,3,['Frank','Mufasa']`
 - `Lx_x_deep` is `[1,2,3,['Frank','Fred']`

Breakout Rooms (3)

- Mutability Activity:
 - Read about and update a scoreboard reporting ranking and team color of contestants.

```
Contestants = [{"name": "fred", "teamColor": "Red"},  
               {"name": "Layla", "teamColor": "Yellow"},  
               {"name": "Tammy", "teamColor": "Green"},  
               {"name": "Buba", "teamColor": "Blue"}]
```

- Your job? Programmatically change the scoreboard as indicated. Hint! Use copy and/or deepcopy if required.

that's it

- Remember that scores & comments are listed in ISVC site.
- If you want to redo assignment 1, go ahead.
- Homeworks are very important - communicate with each other, with the instructors, etc. If you're spending *too* much time, step away and rest ... then tackle with fresh eyes.
- At this point in our studies we might slog thru some code ... There is almost always a more efficient way of doing things and we're going to encounter many them in the coming weeks.

