Computational Structures in Data Science

UC Berkeley EECS
Lecturer
Michael Ball

UC Berkeley EECS
Adj. Ass. Prof.
Dr. Gerald Friedland

Lecture 14:

Mutability

Announcements

- Midterm Grading: Please review your exam.
  - Scores out later today
  - Read now if you want.
  - Regrade requests will OPEN in one week, then you'll have 1 week to submit them.
- Thank you for dealing with the proctoring.
  - We're following up with a few folks about mostly minor issues, but it all went very well!
  - Thanks for those who put comments in the form, were honest and/or just said thanks!

Computing In the News

- Is Everybody Doing … OK? Let's Ask Social Media (NYT)
- Researchers are looking at online behavior to gauge public mental health. The results aren't pretty.
- The Computational Story Lab is part of a small but growing field of researchers who try to parse our national mental health through the prism of our online life
- The researchers call it the Hedonometer. It is the invention of Chris Danforth and his partner Peter Dodds, both trained mathematicians and computer scientists and the co-directors of the lab. The Hedonometer has been up and running for more than a decade now, measuring word choices across millions of tweets, every day, the world over, to come up with a moving measure of well-being.

UC Berkeley EECS
Lecturer
Michael Ball

Computational Structures in Data Science

Mutability: Lists

Learning Objectives

- Distinguish between when a function changes some data, vs returns a new object
- Understand modifying objects in place
- Python provides “is” and “==” for checking if items are the same, in different ways

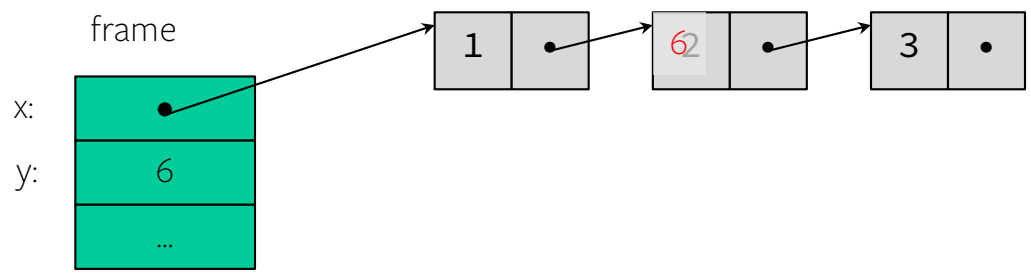
Objects

- An Abstract Data Type consist of data and behavior bundled together to abstract a view on the data
- An object is a concrete instance of an abstract data type.
- Objects can have state
  - mutable vs immutable
  - Mutable: We can change the object after it has been created
  - Immutabe: We cannot change the object.
- Next topic: Object-oriented programming
- In Python, every value is an object
  - All objects have attributes
  - Manipulation happens through method calls

From value to storage …

- A variable assigned a compound value (object) is a reference to that object.
- Mutable object can be changed but the variable(s) still refer to it

```
x = [1, 2, 3]
y = 6
x[1] = y
x[1]
```



Mutation makes sharing visible

Python 3.6

```
1  x = 2
2  y = 3
3  print(x+y)
4  x = 4
5  print(x+y)
```

Edit this code

Print output (drag lower right corner to resize)

```
5
7
```

Frames | Objects

Global frame
x 4
y 3

Python 3.6

```
1  x = [1, 2, 3]
2  y = x
3  print(y)
4  x[1] = 11
5  print(y)
```

Edit this code

Print output (drag lower right corner to resize)

```
[1, 2, 3]
[1, 11, 3]
```

Frames | Objects

Global frame
x
y

list
0 1 2
1 11 3

Mutating Input Data

- Functions can mutate objects passed in as an argument
- Declaring a new variable with the same name as an argument only exists within the scope of our function
  - You can think of this as creating a new name, in the same way as redefining a variable.
  - This will not modify the data outside the function, even for mutable objects.
- BUT, we can still directly modify the object passed in…even though it was created in some other frame or environment.
  - We directly call methods on that object.
- View Python Tutor

Copies, ‘is’ and ‘==‘

```
>>> alist = [1, 2, 3, 4]
>>> alist == [1, 2, 3, 4]  # Equal values?
True
>>> alist is [1, 2, 3, 4]  # same object?
False
>>> blist = alist          # assignment refers
>>> alist is blist         # to same object
True
>>> blist = list(alist)    # type constructors copy
>>> blist is alist
False
>>> blist = alist[ : ]     # so does slicing
>>> blist is alist
False
>>> blist
[1, 2, 3, 4]
>>>
```

UC Berkeley EECS
Lecturer
Michael Ball

Computational Structures in Data Science

Mutable Functions

Learning Objectives

- Distinguish between when a function changes some data, vs returns a new object
- Understand modifying objects in place
- global allows a function to modify a global variable.
- nonlocal allows a function to modify a variable in an outer scope, such as a parent frame
  - But does not look in the global frame even if it is the parent

Creating mutating 'functions'

- Pure functions have referential transparency
  - `c = greet() + name()` is "referentially transparent" if we can replace that expression with the value, maybe that's `c = "Hello, CS 88"`
  - Result value depends only on the inputs
    - Same inputs, same result value
- Functions that use global variables are not pure
- They can be "mutating", and rely on some state

```
>>> counter = 0
>>> def count_fun():
...     global counter
...     counter += 1
...     return counter
...
>>> count_fun()
1
>>> count_fun()
2
```

Creating mutating 'functions'

```
>>> counter = 0
>>> def count_fun():
...     global counter
...     counter += 1
...     return counter
...
>>> count_fun()
1
>>> count_fun()
2
```

How do I make a second counter?

```
>>> def make_counter():
...     counter = 0
...     def counts():
...         nonlocal counter
...         counter +=1
...         return counter
...     return counts
...
>>> count_fun = make_counter()
>>> count_fun()
1
>>> count_fun()
2
>>> another_one = make_counter()
>>> another_one()
1
>>> count_fun()
3
```