Data aliasing

Terence Parr
MSDS program
University of San Francisco

See notebook: https://github.com/parrt/msds501/blob/master/notes/aliasing.ipynb

UNIVERSITY OF SAN FRANCISCO

Variables refer to memory regions

- We use names like **data** and **salary** to represent memory cells holding data values
- The names are easier to remember than the physical memory addresses, but we can get fooled
- Variables are really *references* or *pointers* to chunks of memory
- Pointers are like phone numbers that "point at" phones, but phone numbers are not the phone itself

Uncovering memory locations

- Two variables x and y can both have the same value 7, but they are technically both pointing to the same 7 object!
- We can uncover this secret level of indirection using the built-in **id(x)** function that returns the physical memory address pointed to by **x**

```
x = y = 7
print(x,y)
```

```
7 7
```

```
x = y = 7
print(id(x))
print(id(y))
```

```
4561599008
4561599008
```

Aliasing: variables pointing at the same memory region

- Assigning one variable to another creates an *alias* because both variables now point at the same memory location

```
name = 'parrt'
userid = name # userid now points at the same memory as name
print(id(name), id(userid))
```

```
140404363692336 140404363692336
```

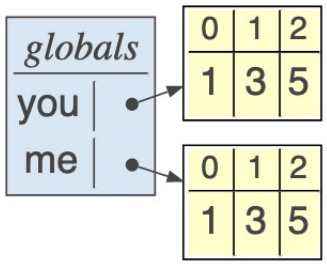
- That memory location contains the five letters p-a-r-r-t
- **name** and **userid** look like two copies, but they share the same location in memory

Altering non-overlapping data

- Each […] list literal creates a new list even if the elements within the lists are the same

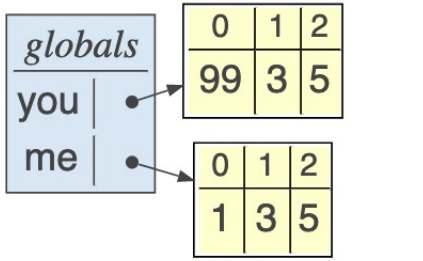
```
you = [1,3,5]
me  = [1,3,5]
print(id(you), id(me))
```

```
140404909444608 140404909444544
```




```
you = [1,3,5]
me  = [1,3,5]
print(you, me)
you[0] = 99
print(you, me)
```

```
[1, 3, 5] [1, 3, 5]
[99, 3, 5] [1, 3, 5]
```

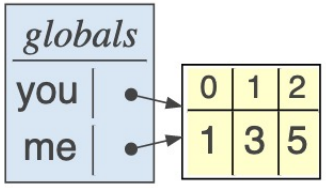


Altering shared (aliased) data

- If we make **you** and **me** share the same list (same region of memory), however, then changing one changes the other!!!

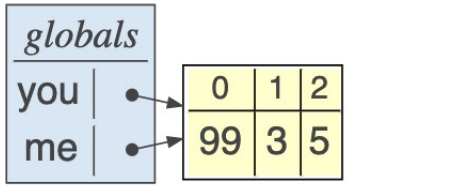
```
you = [1,3,5]
me  = you
print(id(you), id(me))
print(you, me)
```

```
140404908360448 140404908360448
[1, 3, 5] [1, 3, 5]
```




```
you[0] = 99
print(you, me)
```

```
[99, 3, 5] [99, 3, 5]
```



Experiment in pythontutor

Reassigning a var breaks the aliasing

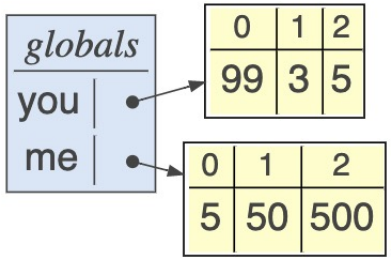
- Don't confuse changing the pointer to a list with changing the list elements:

```
you = [1,3,5]
me = you
me[0] = 99
me = [5,50,500]
print(me,you)
```

Changes 'you'

Doesn't change 'you'

```
[5, 50, 500] [99, 3, 5]
```



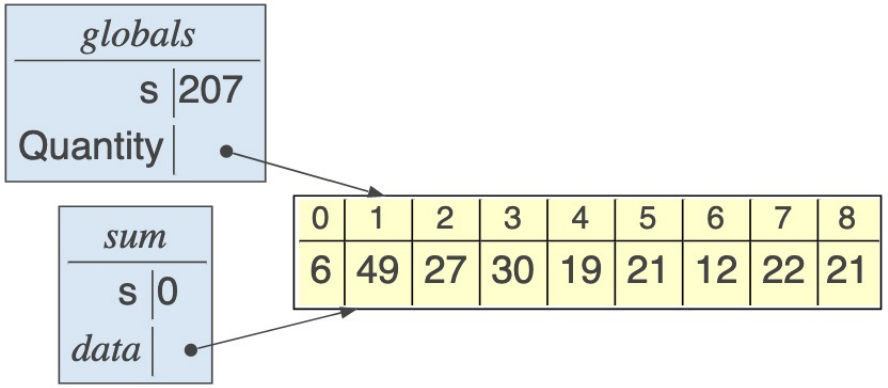
(We'll learn about functions in more detail soon)

Aliasing through argument passing

- Aliasing of data happens a great deal when we pass lists or other data structures to functions
- E.g., passing list **Quantity** to a function whose argument is called **data** means that the two are aliased

```
def sum(data):
    s = 0
    for q in data:
        s = s + q
    return s

Quantity = [6, 49, 27, 30, 19, 21, 12, 22, 21]
sum(Quantity)
```

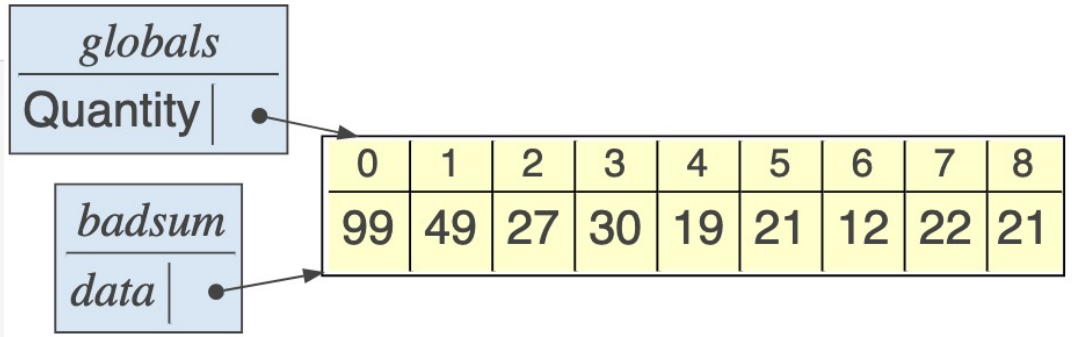


Warning: Functions can alter the contents of aliased arguments (preview of funcs)

- If **badsum**() alters the argument, it also alters the global since they point at the same data

```
def badsum(data):
    data[0] = 99
    s = 0
    for q in data:
        s = s + q
    return s

Quantity = [6, 49, 27, 30, 19, 21, 12, 22, 21]
print(Quantity)
badsum(Quantity)
print(Quantity)
```




```
[6, 49, 27, 30, 19, 21, 12, 22, 21]
[99, 49, 27, 30, 19, 21, 12, 22, 21]
```