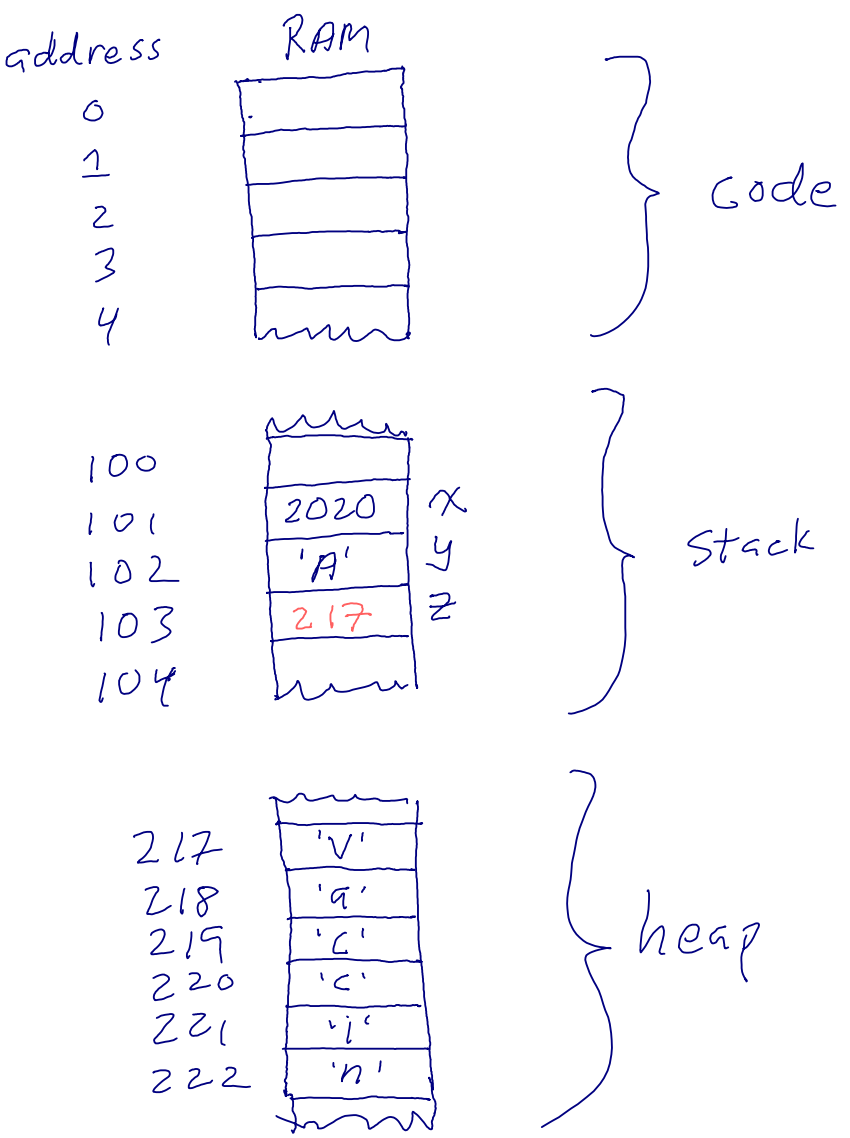
The truth about data representation and how to depict it

```
x = 2020
y = 'A'
z = 'vaccine'
w = [1,2,3]
```

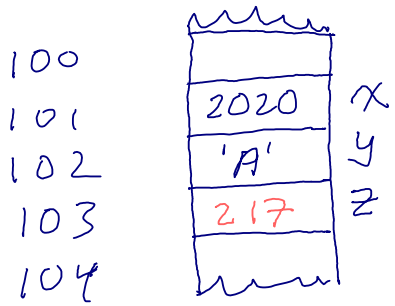
} don't fit one word of memory

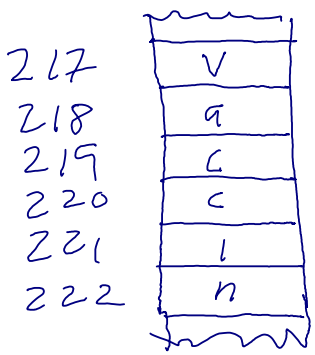


Textbook section 5.4

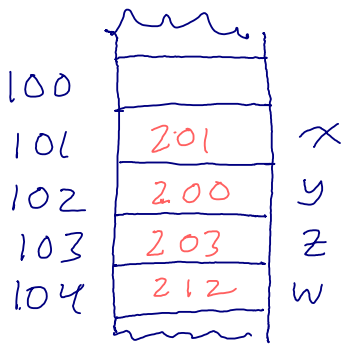
For uniformity, Python variables all hold references.

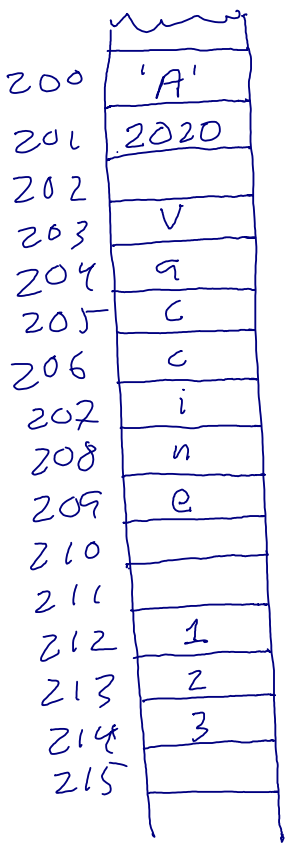
not this



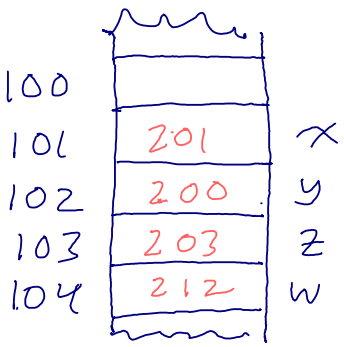


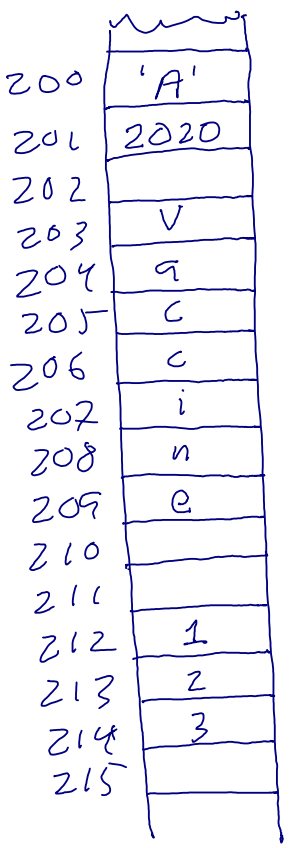
but this

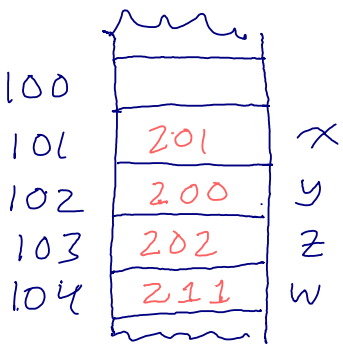


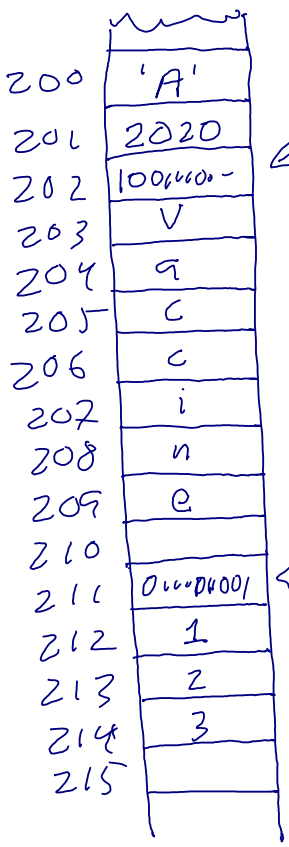


Full disclosure: there are actually object descriptors too, so it's really like this:







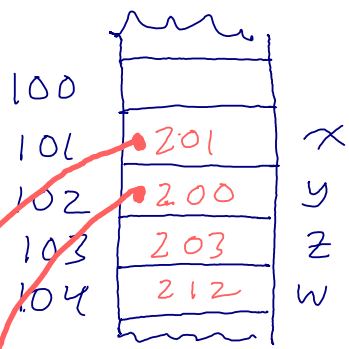


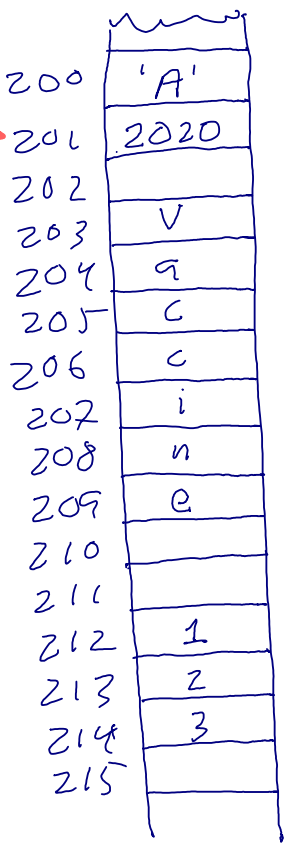
binary code for here is a string of length 7

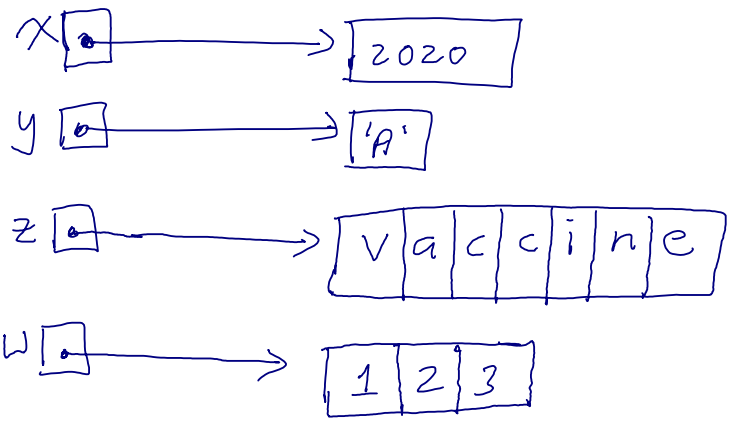
binary code for here is a list of length 3

But we can safely ignore that.

The specific addresses don't matter so we use box-and-arrow diagrams.







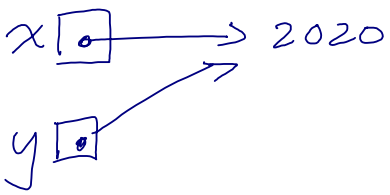
To see some actual addresses try this in IDLE:

```
x = 2020
y = 'A'
z = 'vaccine'
x
x+1
id(x)
id(y)
```

} implicitly dereference to get value

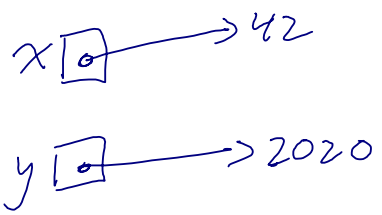
Assignment copies one reference

```
x = 2020
y = x
```



and then

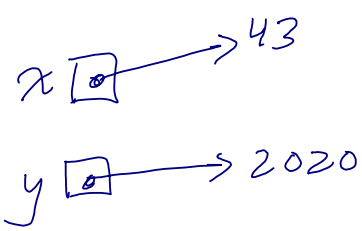
```
x = 42
```



and then

```
x = x + 1
```

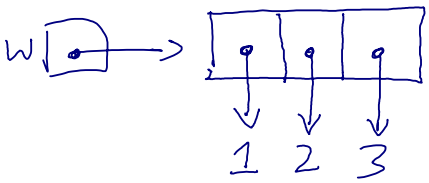
on right side of assignment, dereference x to get its value



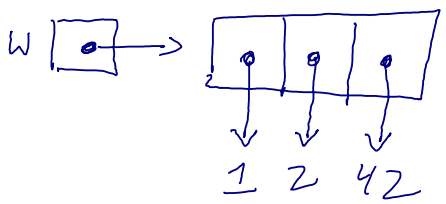
42

Lists are mutable — contents are ref's

```
w = [1, 2, 3]
```




```
w[2] = 42
```

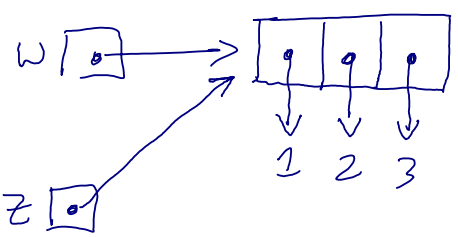


Punch line — pain point for programmers

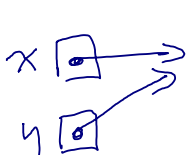
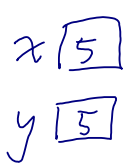
```
w = [1, 2, 3]

z = w

w[2] = 42
print (z[2])
```



Consolation: immutables can't change

so  is no trouble and we can just write it as 

After `y = 6` we get 

Same with strings — and immutables <u>in</u> lists.