

The truth about data representation and how to depict it ①

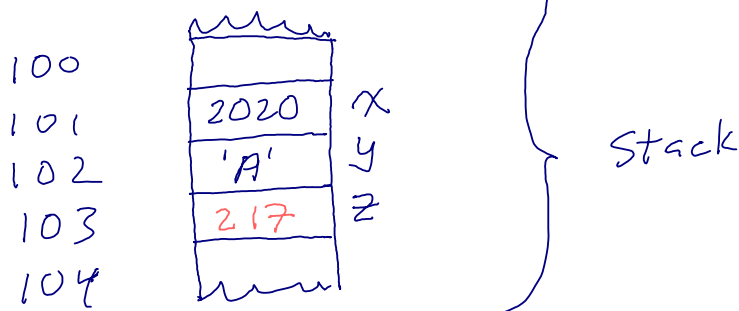
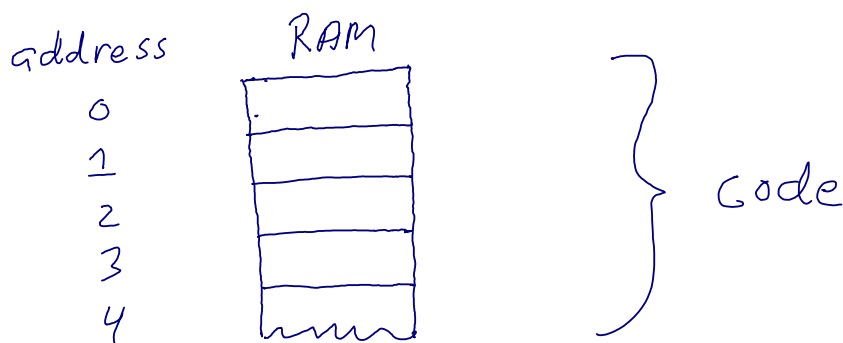
$x = 2020$

$y = 'A'$

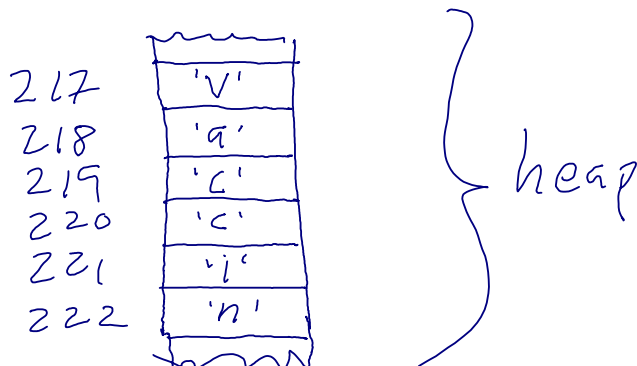
$z = 'vaccine'$

$w = [1, 2, 3]$

} don't fit one word of memory



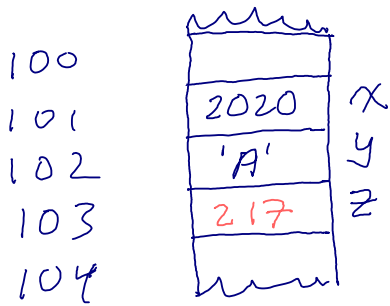
value of z is a
reference to
elsewhere in mem.



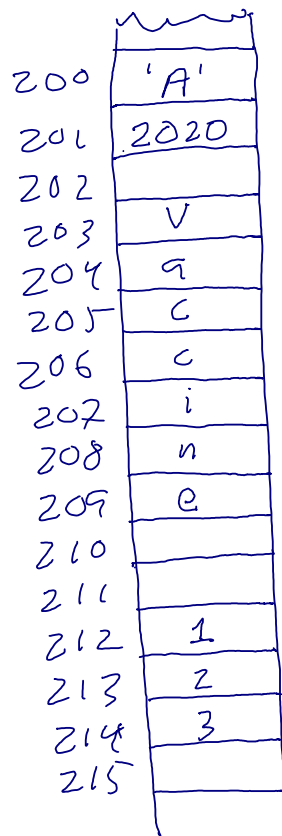
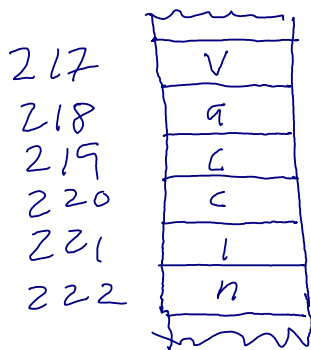
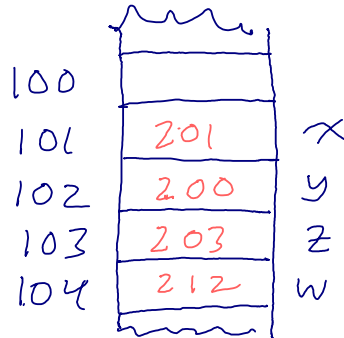
Textbook section 5.4

For uniformity, Python variables all hold references. ②

not this



but this



Full disclosure:

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

③

100		
101	201	x
102	200	y
103	203	z
104	212	w

100		
101	201	x
102	200	y
103	202	z
104	211	w

200	'A'
201	2020
202	
203	V
204	a
205	c
206	c
207	i
208	n
209	e
210	
211	
212	1
213	2
214	3
215	

200	'A'
201	2020
202	10011001
203	V
204	a
205	c
206	c
207	i
208	n
209	e
210	
211	01111001
212	1
213	2
214	3
215	

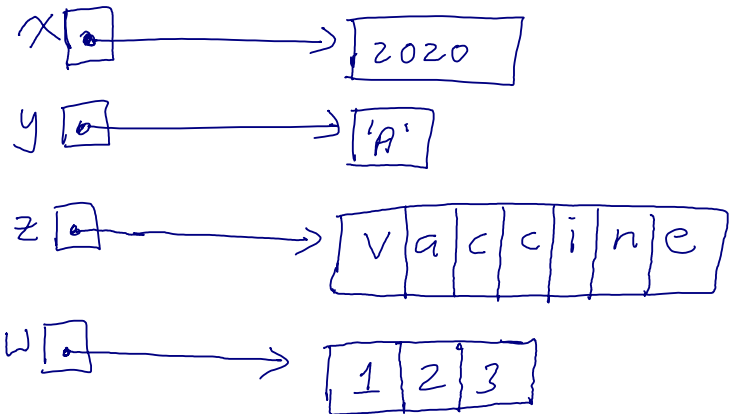
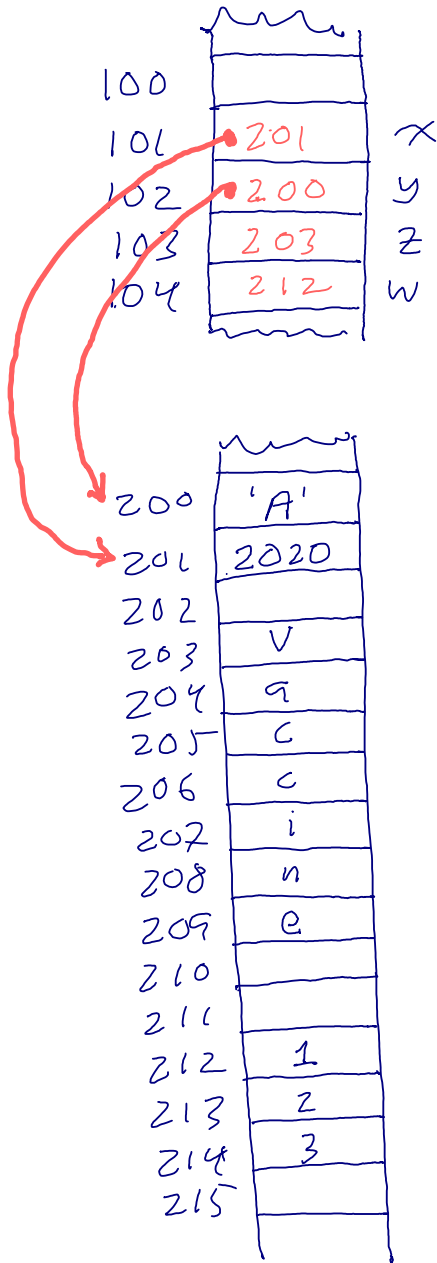
← binary code for here is a string of length 7

← binary code for here is a list of length 3

But we can safely ignore that.

(4)

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
 $x+1$
 $id(x)$
 $id(y)$

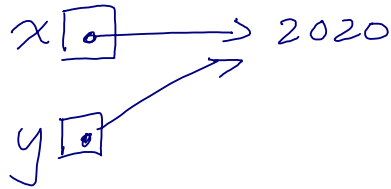
} implicitly dereference to get value

⑤

Assignment copies one reference

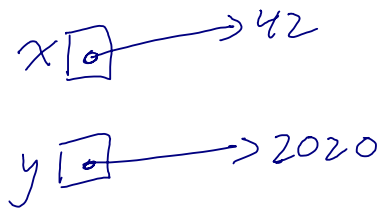
$x = 2020$

$y = x$



and then

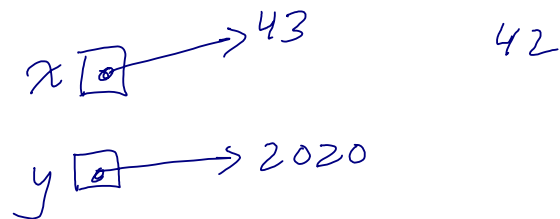
$x = 42$



and then

$x = x + 1$

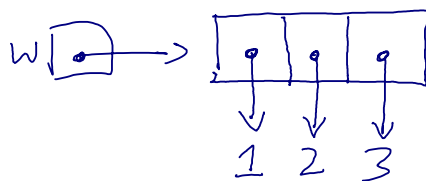
↑
on right side
of assignment,
dereference x
to get its value



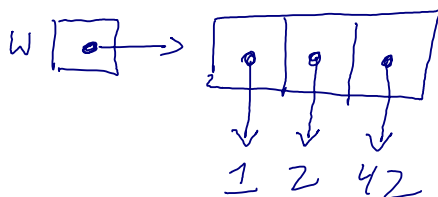
⑥

Lists are mutable - contents are ref's

$w = [1, 2, 3]$

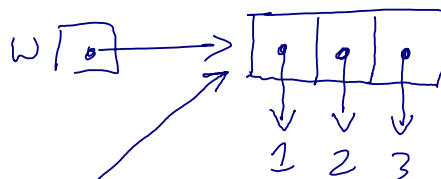


$w[2] = 42$



Punch line — pain point for programmers

$w = [1, 2, 3]$



$z = w$



$w[2] = 42$

print ($z[2]$)

Consolation: immutables can't change

so $x \rightarrow 5$ is no trouble and we can just write it as $x[5]$
 $y \rightarrow 5$

After $y = 6$ we get $x[5]$
 $y[6]$

Same with strings — and immutables in lists.

