doesn't require module installation using 'pip'

doesn't require import statement

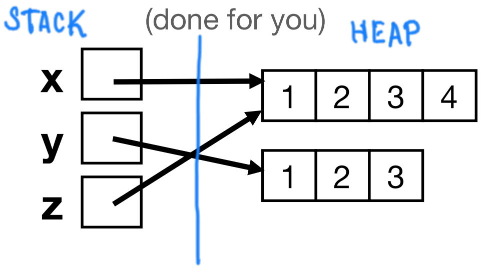
Put True (T) or False (F) in every cell, based on characteristics of each type.

1

Data Type	Mutable?	Pre-installed?	Builtin?	Create New Types?	Named Attributes?
list	T	T	T	F	F
tuple	F	T	T	F	F
namedtuple	F	T	F	T	T ex: p.age, p.fname,...

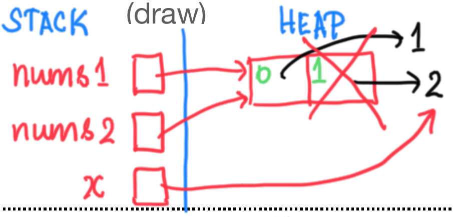
2

```
x = [1, 2, 3]
y = [1, 2, 3]
z = x
z.append(4)
```



3

```
nums1 = [1,2]
nums2 = nums1
x = nums2.pop(1)
```

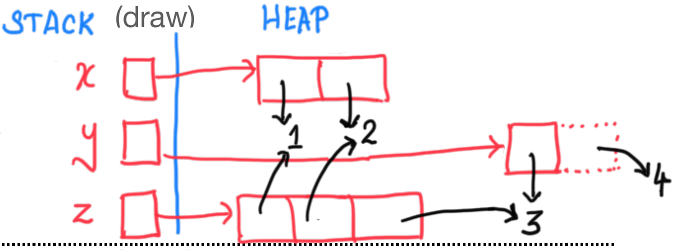


4

```
x = [1, 2]
y = [3]
z = x + y
y.append(4)
```

Recall that '+' operator creates a new object instance.

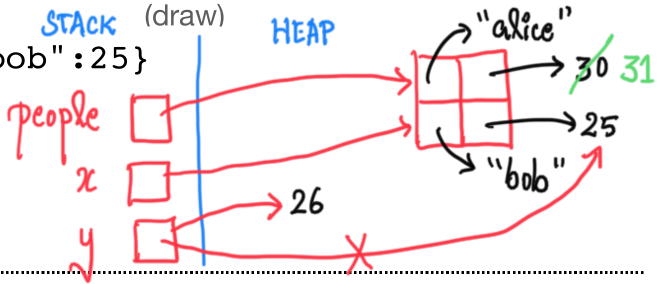
only appends 4 to the object referenced by y



5

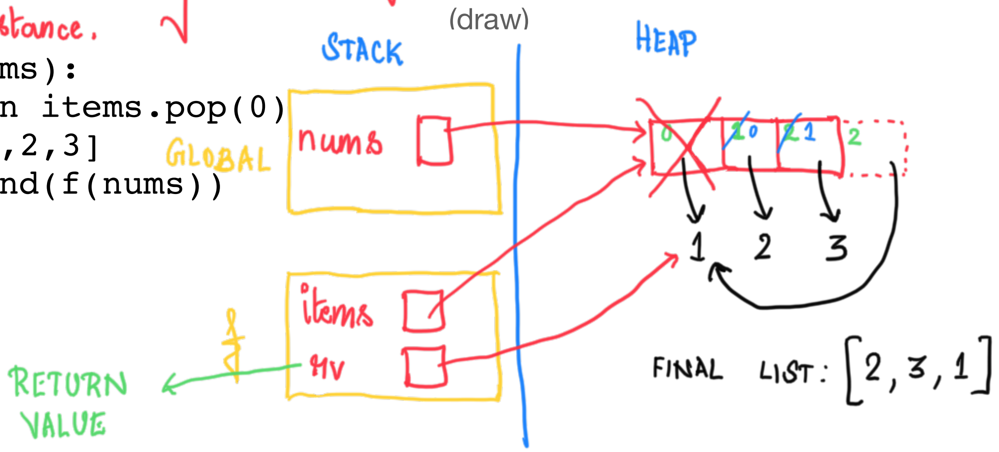
```
people = {"alice":30, "bob":25}
x = people
y = people["bob"]
x["alice"] = 31
y = 26
```

only modifies y's reference. Does not change dict object instance.



6

```
def f(items):
    return items.pop(0)
nums = [1,2,3]
nums.append(f(nums))
```



FINAL LIST: [2, 3, 1]

SHALLOW COPY: creates copy of object at DEPTH LEVEL 1

This is a mistake.

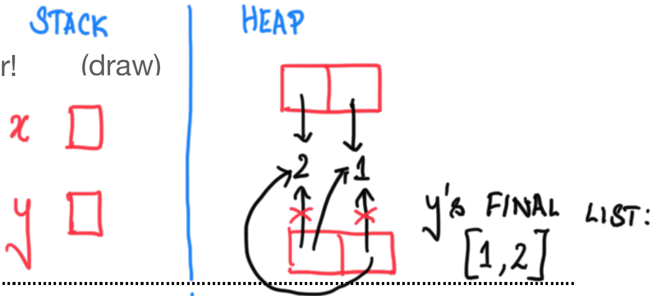
DEEP COPY: creates copies of objects at all DEPTH LEVELS

REFERENCE COPY: no new object instances

Remember to import copy for these in Python Tutor!

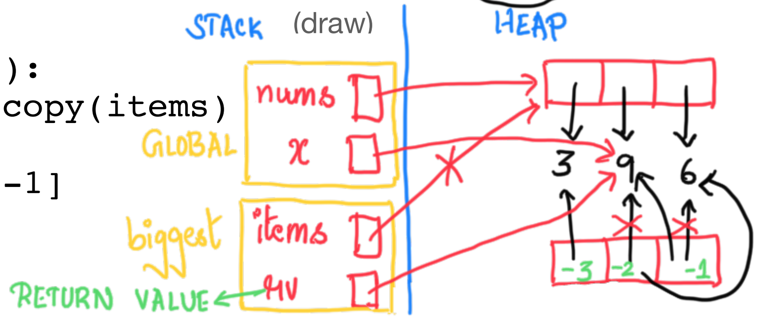
7

```
x = [2,1]
y = copy.copy(~~y~~x)#shallow copy
y.sort()#in-place sort affects original object instance
```



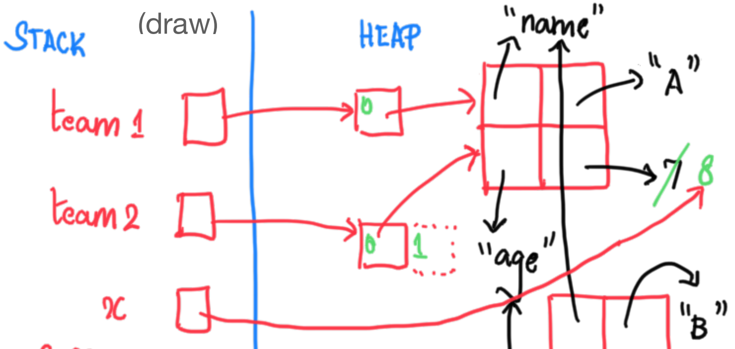
8

```
def biggest(items):
    items = copy.copy(items)
    items.sort()
    return items[-1]
nums = [3,9,6]
x = biggest(nums)
```



9

```
team1 = [
  {"name":"A", "age":7}
]
team2 = copy.copy(team1)
team2.append(
  {"name":"B", "age":9}
)
team2[0]["age"] = 8
x = team1[0]["age"]#8 for shallow copy
```

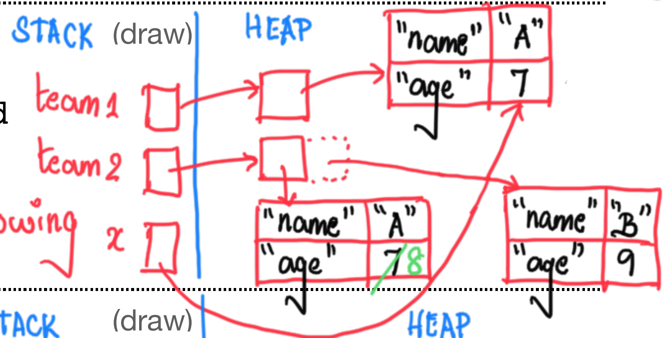


7 for deepcopy

10

Same as above, but with copy.deepcopy(...) instead of copy.copy(...).

Simplifying visualization by showing primitive objects inline

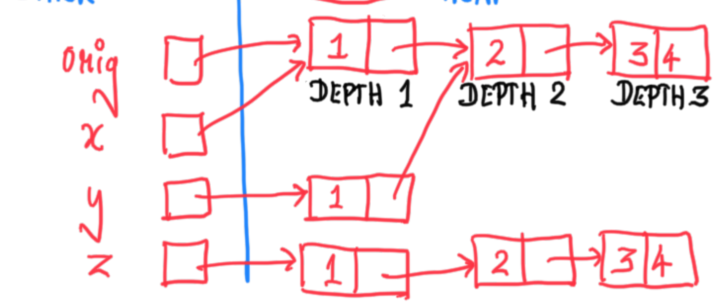


11

```
orig = [1,[2,[3,4]]]
x = orig
y = copy.copy(orig)
z = copy.deepcopy(orig)
```

SHALLOW COPY

DEEP COPY


STACK
(draw)
HEAP
orig
x
y
z
1
2
3
4
DEPTH 1
DEPTH 2
DEPTH 3
1
1
2
3
4