Python WATs: Uncovering Odd Behavior

Who am I?

Hacker School Alum
Software Engineer at Venmo

@amygdalama
mathamy.com

Thanks!

Allison Kaptur
Tom Ballinger
Gary Bernhardt

Themes

Identity
Mutability
Scope

```
for theme in themes:
```

a. Trivia

b. Why?

c. Tips

Identity

Trivia

```
> "" is ""
???
```

Trivia

```
> "" is ""
True
```

Trivia

```
> 0 is 0
???
```

Trivia

```
> 0 is 0
True
```

Trivia

```
> [] is []
???
```

Trivia

```
> [] is []
False
```

Trivia

```
> {} is {}
???
```

Trivia

```
> {} is {}
False
```

Trivia

```
> a = 256
> b = 256
> a is b
???
```

Trivia

```
> a = 256
> b = 256
> a is b
True
```

Trivia

```
> a = 257
> b = 257
> a is b
???
```

Trivia

```
> a = 257
> b = 257
> a is b
False
```

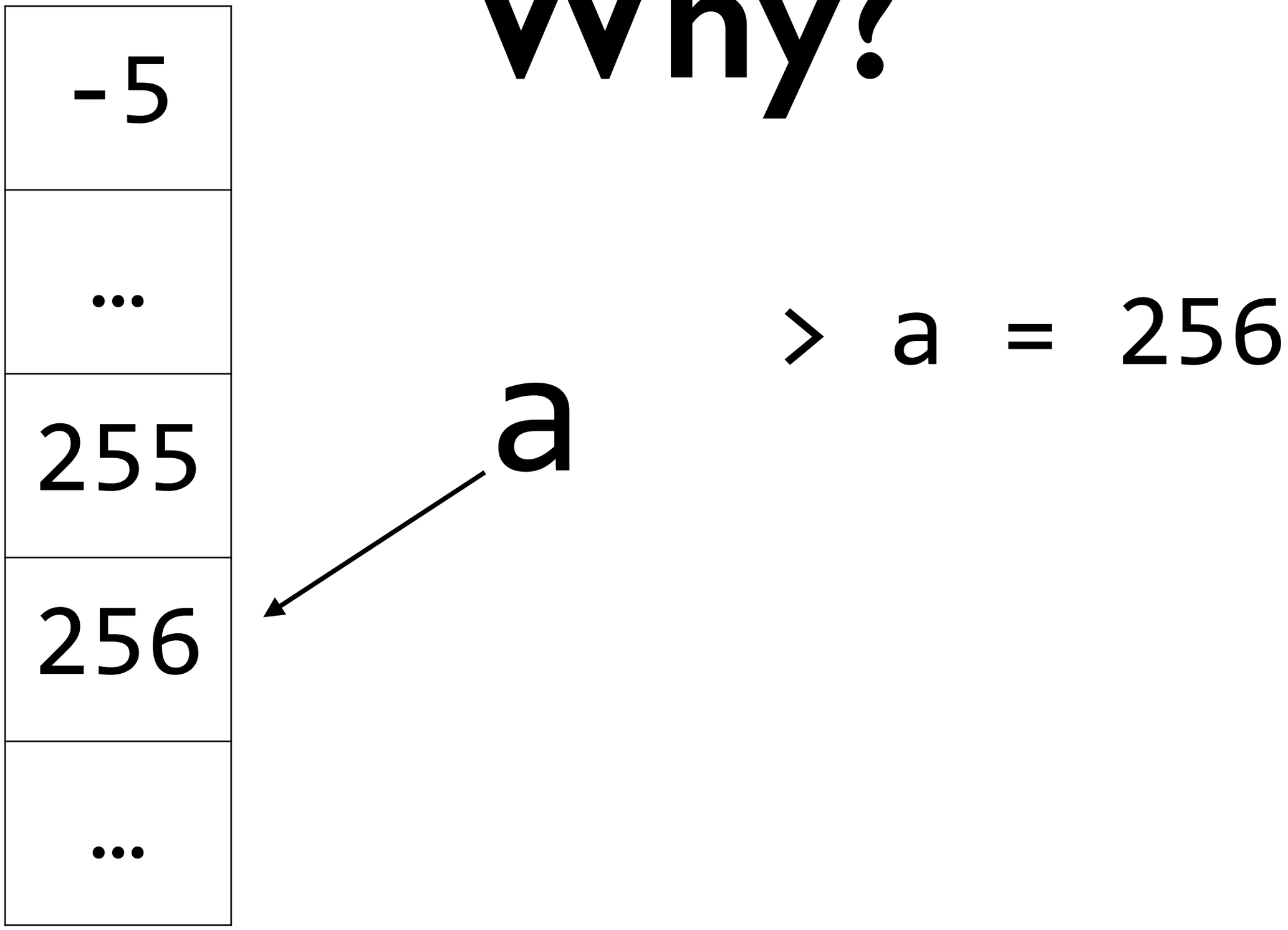
Trivia

```
> a = 257; b = 257
> a is b
???
```

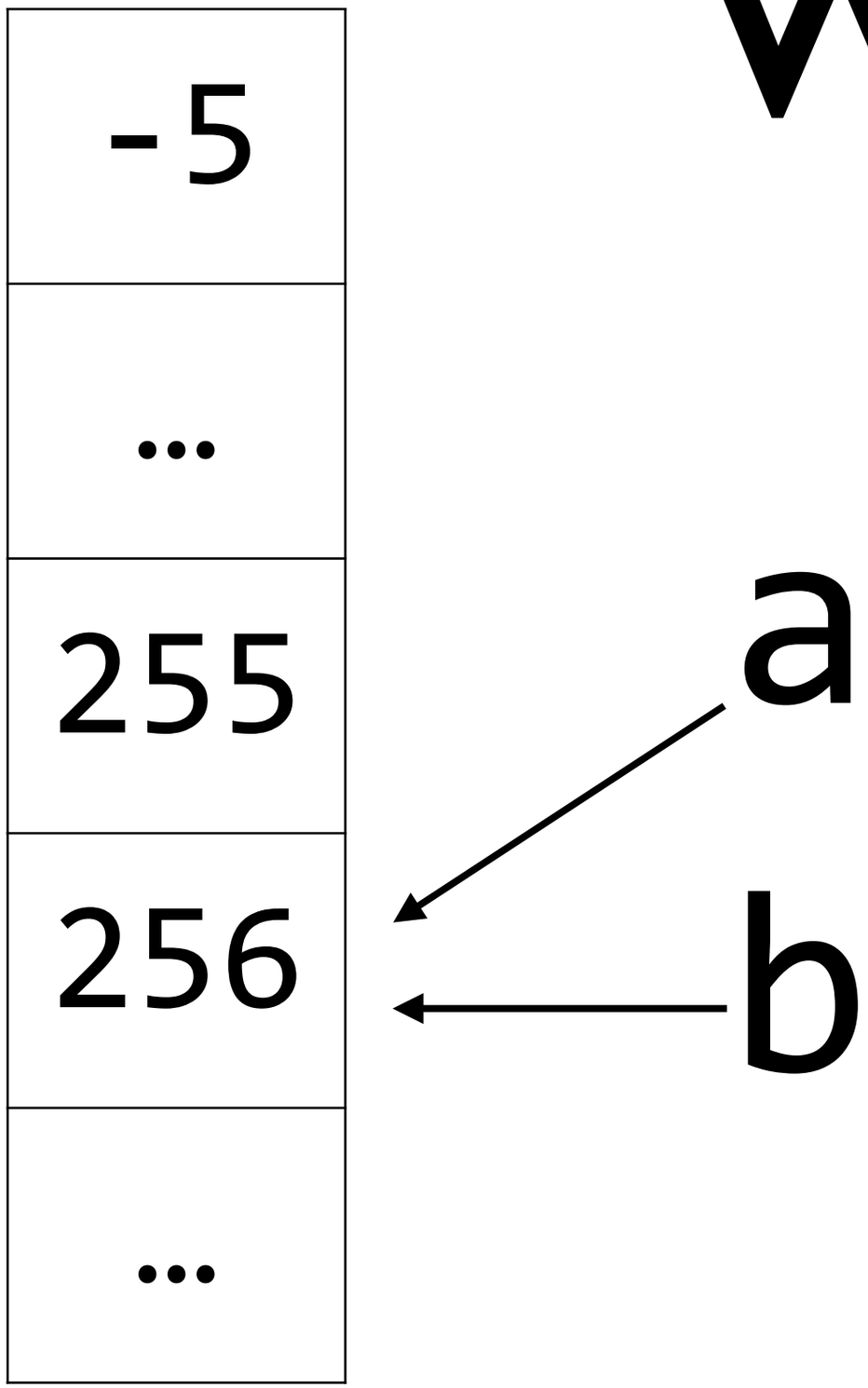
Trivia

```
> a = 257; b = 257
> a is b
True
```

Why?

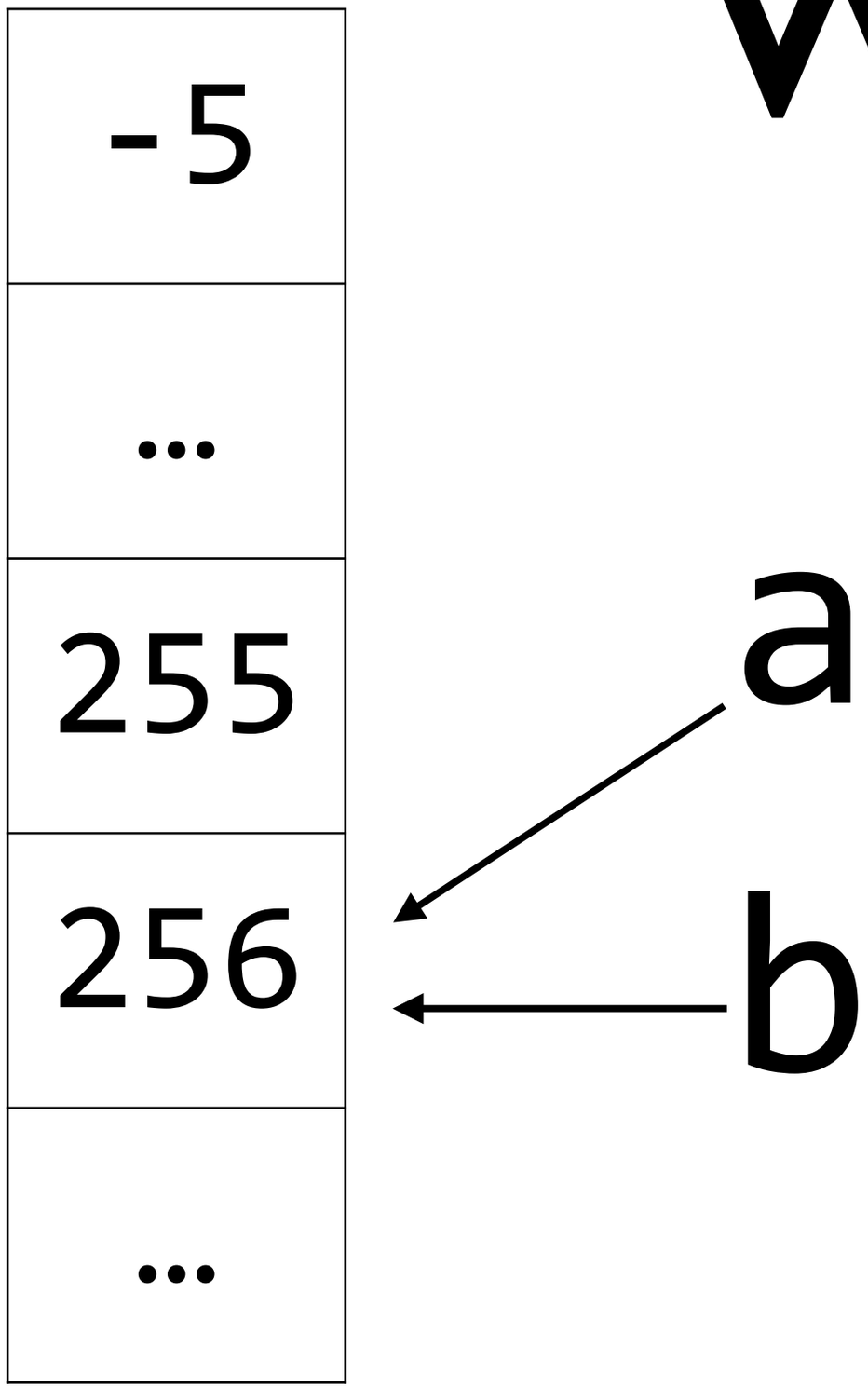


Why?



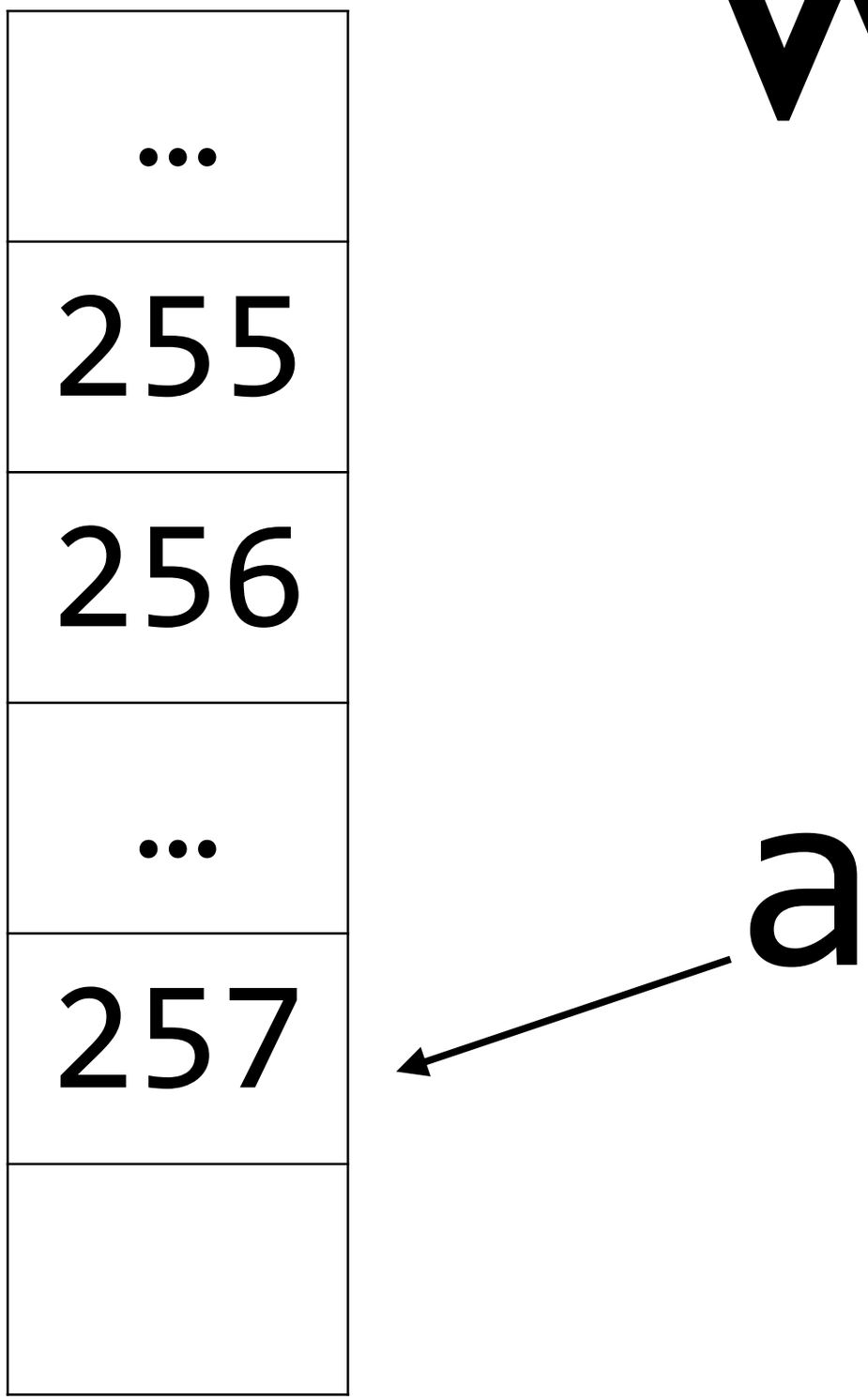

```
> a = 256
> b = 256
```

Why?



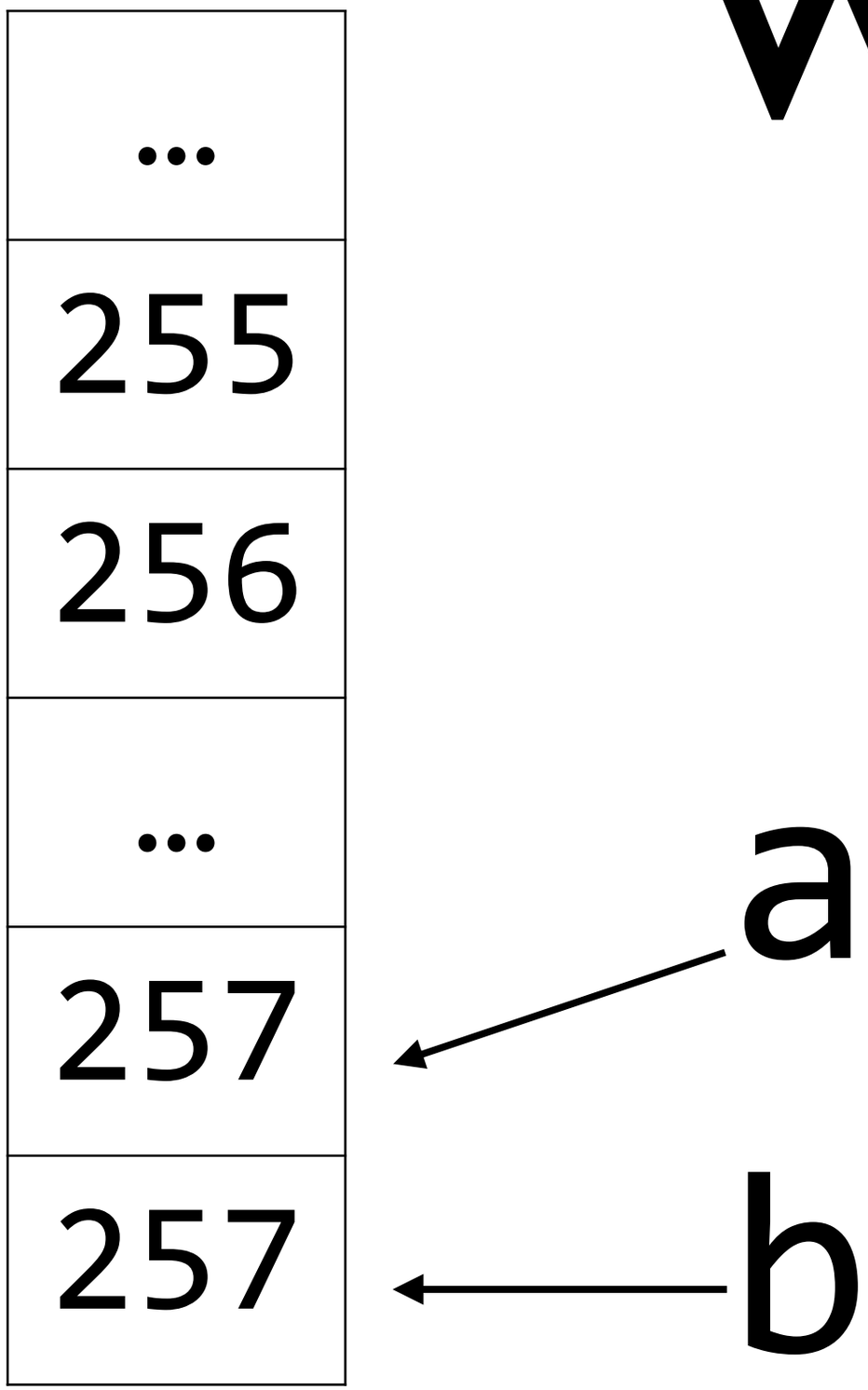

```
> a = 256
> b = 256
> a is b
True
```

Why?



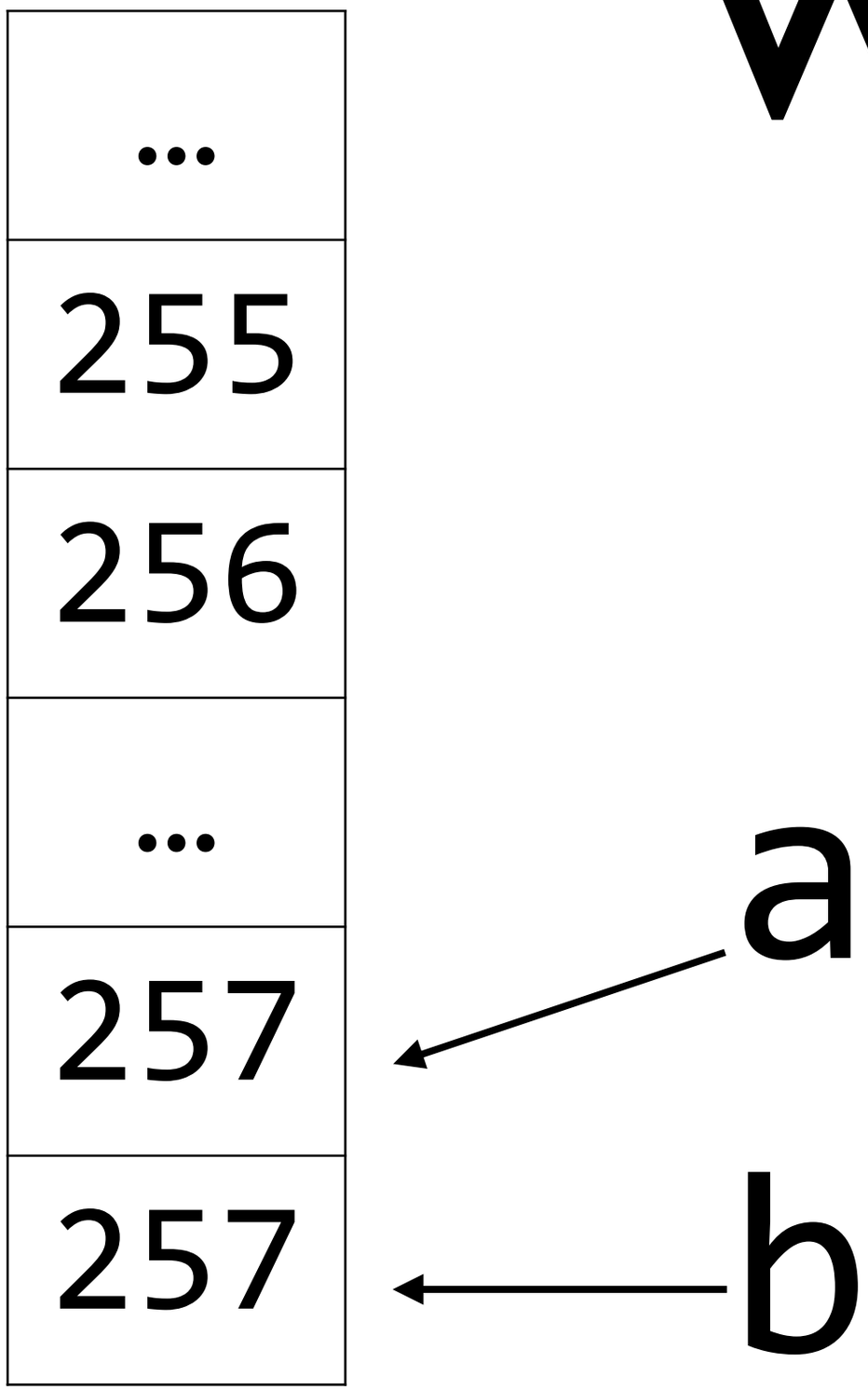

```
> a = 257
```

Why?




```
> a = 257
> b = 257
```

Why?




```
> a = 257
> b = 257
> a is b
False
```

Why?

```
> a = 257
> b = 257
> id(a) == id(b)
False
```

Why?

```
> a = 257; b = 257
> id(a) == id(b)
True
```

Tips

- Use `==` instead of `is` to compare values
- Exceptions:
  - `x is None`
  - tests for identity

Mutability

Trivia

```
> faves = ["cats", "dragons"]
```

Trivia

```
> faves = ["cats", "dragons"]
> temp = faves
```

Trivia

```
> faves = ["cats", "dragons"]
> temp = faves
> temp.append("rainbows")
```

Trivia

```
> faves = ["cats", "dragons"]
> temp = faves
> temp.append("rainbows")
> temp
???
```

Trivia

```
> faves = ["cats", "dragons"]
> temp = faves
> temp.append("rainbows")
> temp
["cats", "dragons", "rainbows"]
```

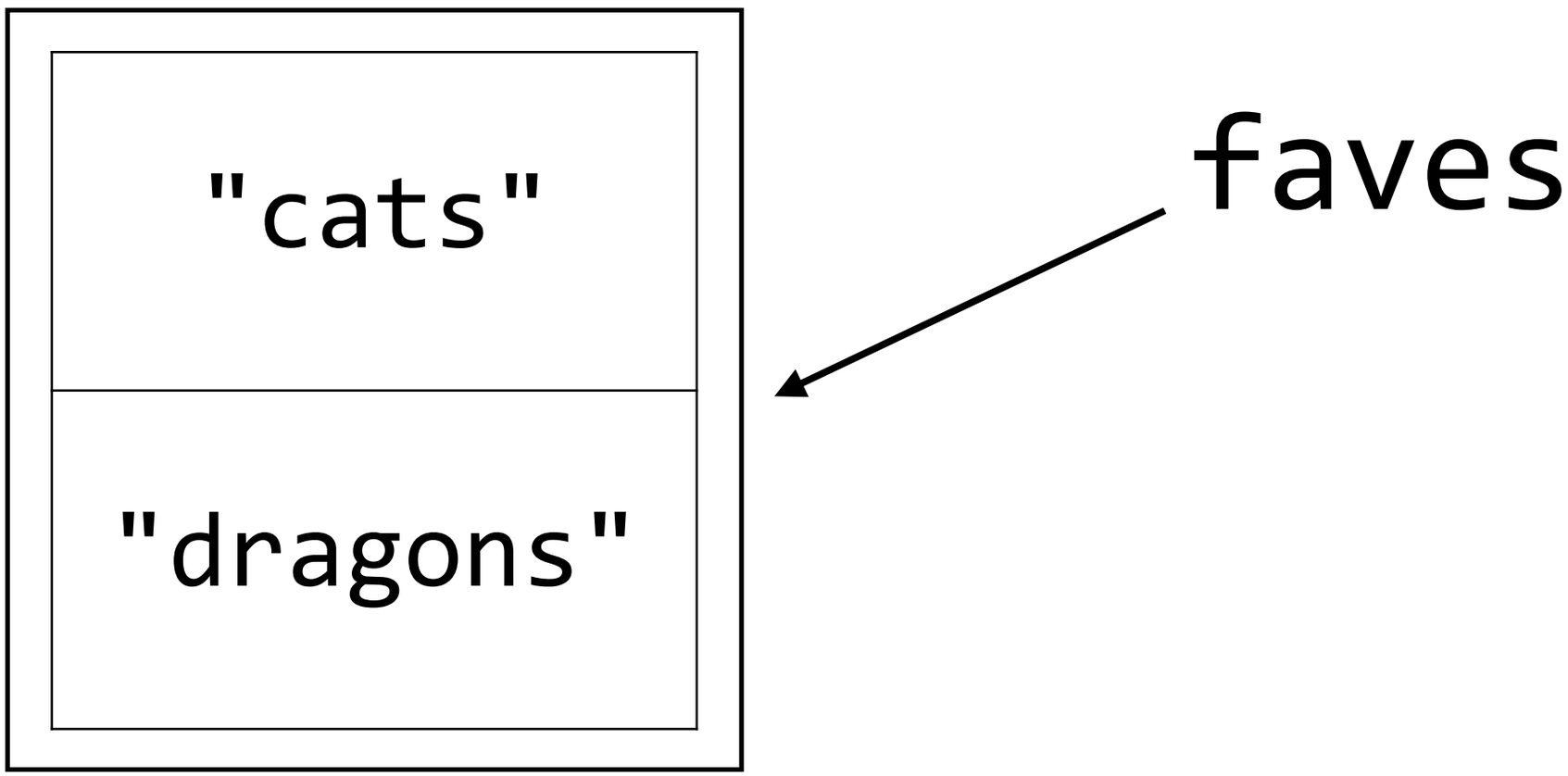
Trivia

```
> faves = ["cats", "dragons"]
> temp = faves
> temp.append("rainbows")
> faves
???
```

Trivia

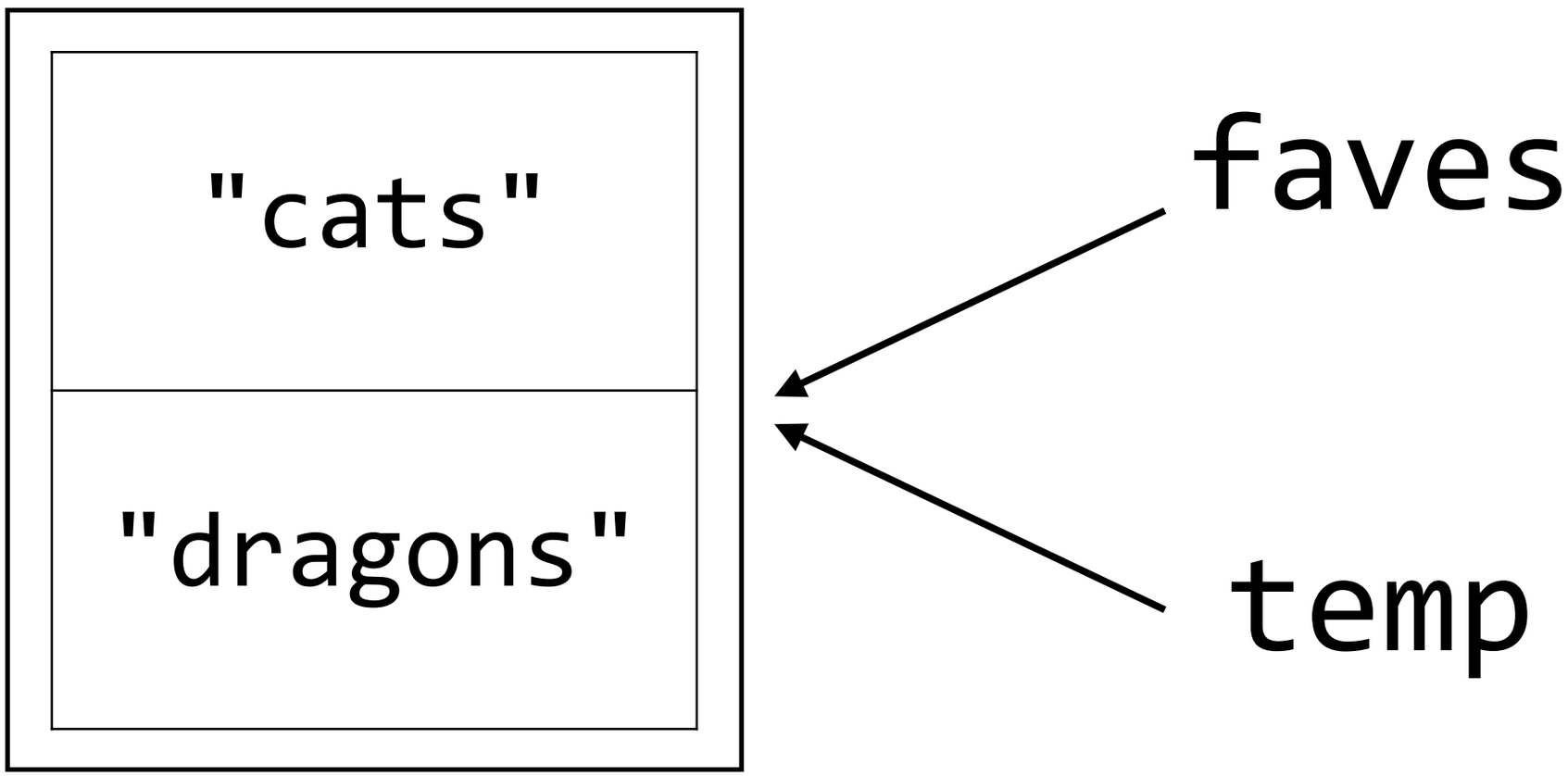
```
> faves = ["cats", "dragons"]
> temp = faves
> temp.append("rainbows")
> faves
["cats", "dragons", "rainbows"]
```

Why?



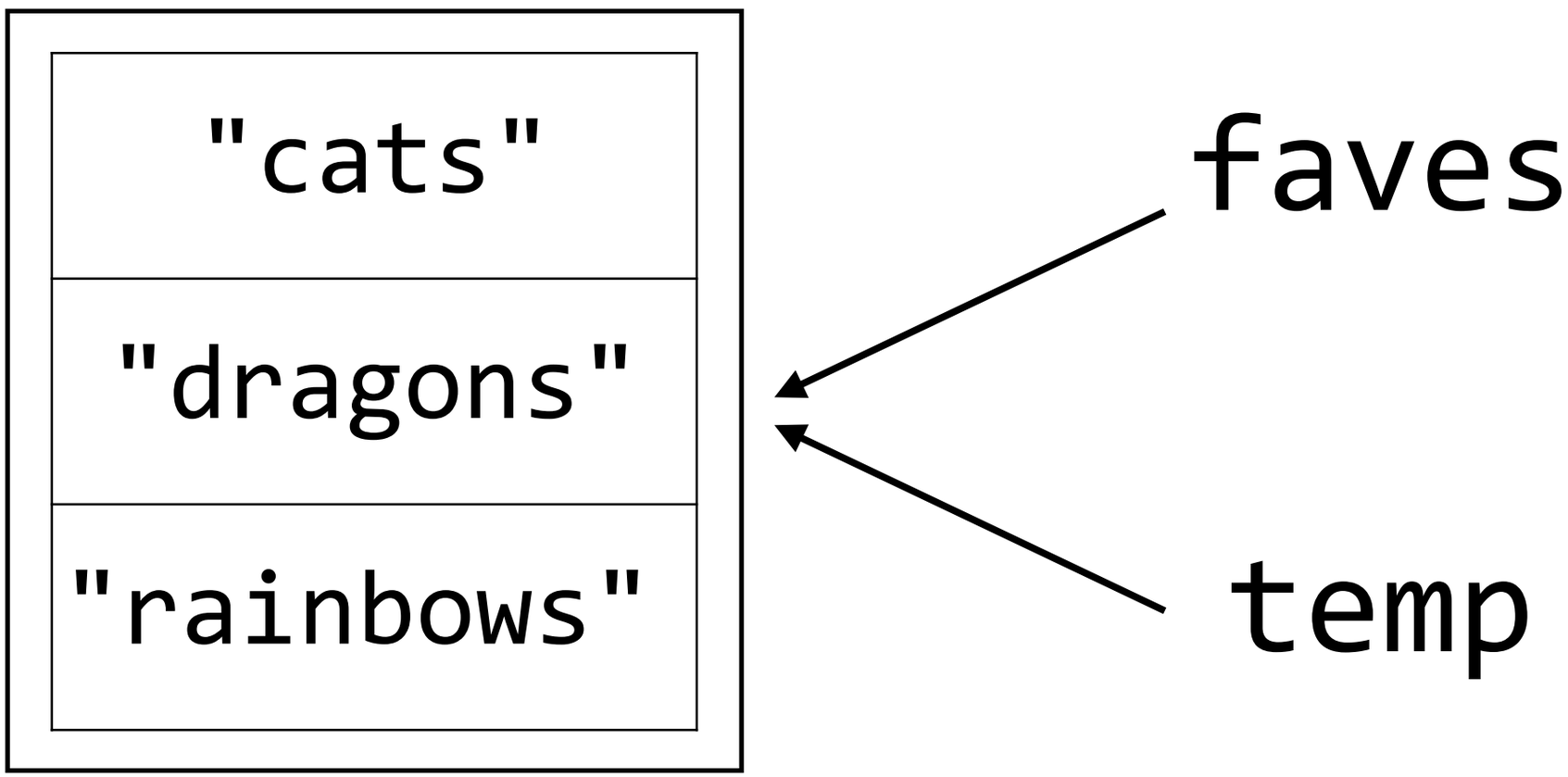

```
> faves = ["cats", "dragons"]
```

Why?




```
> temp = faves
```

Why?




```
> temp.append("rainbows")
```

Tip: Mutable Objects

- lists [ ]
- dictionaries { }
- sets ( )

Tip: Make Real Copies

```
> temp = faves[:]
> temp = list(faves)
```

Tip: Shallow Copy

```
> temp = []
> for element in faves:
      temp.append(element)
```

Tip: Deep Copy

```
> faves = [["cats", "dragons"]]
```

Tip: Deep Copy

```
> faves = [["cats", "dragons"]]
> temp = faves[:]
```

Tip: Deep Copy

```
> faves = [["cats", "dragons"]]
> temp = faves[:]
> temp[0].append("rainbows")
```

Tip: Deep Copy

```
> faves = [["cats", "dragons"]]
> temp = faves[:]
> temp[0].append("rainbows")
> faves
???
```

Tip: Deep Copy

```
> faves = [["cats", "dragons"]]
> temp = faves[:]
> temp[0].append("rainbows")
> faves
[["cats", "dragons", "rainbows"]]
```

Tip: Deep Copy

- For arbitrarily-nested lists, make deep copies

```
> import copy
> temp = copy.deepcopy(faves)
```

Mutable Default Arguments

Trivia

```
def append_cat(l=[]):
```

Trivia

```
def append_cat(l=[]):
    l.append('cat')
    return l
```

Trivia

```
def append_cat(l=[]):
    l.append(‘cat’)
    return l

> append_cat()
???
```

Trivia

```
def append_cat(l=[]):
    l.append(‘cat’)
    return l

> append_cat()
[‘cat’]
```

Trivia

```
def append_cat(l=[]):
    l.append('cat')
    return l

> append_cat()
['cat']
> append_cat()
???
```

Trivia

```
def append_cat(l=[]):
    l.append('cat')
    return l

> append_cat()
['cat']
> append_cat()
['cat', 'cat']
```

Why?

```
def append_cat(l=[]):
    l.append('cat')
    return l

> append_cat.func_defaults
???
```

Why?

```
def append_cat(l=[]):
    l.append(‘cat’)
    return l

> append_cat.func_defaults
([],)
```

Why?

```
def append_cat(l=[]):
    l.append('cat')
    return l

> append_cat()
> append_cat.func_defaults
???
```

Why?

```
def append_cat(l=[]):
    l.append('cat')
    return l

> append_cat()
> append_cat.func_defaults
(['cat'],)
```

Why?

```
> append_cat.func_defaults
(['cat'],)
```

Why?

```
> append_cat.func_defaults
(['cat'],)
> _[0].append('dragon')
```

Why?

```
> append_cat.func_defaults
(['cat'],)
> _[0].append('dragon')
> append_cat()
???
```

Why?

```
> append_cat.func_defaults
(['cat'],)
> _[0].append('dragon')
> append_cat()
['cat', 'dragon', 'cat']
```

Tip: Use None

```
def append_cat(l=None):
```

Tip: Use None

```
def append_cat(l=None):
    if l is None:
        l = []
```

Tip: Use None

```
def append_cat(l=None):
    if l is None:
        l = []
    l.append(‘cat')
    return l
```

Tip: Take Advantage!

```
def sorting_hat(student, cache={}):
```

Tip: Take Advantage!

```
def sorting_hat(student, cache={}):
    if student in cache:
        print "Used cache!"
        return cache[student]
    else:
```

Tip: Take Advantage!

```
def sorting_hat(student, cache={}):
    if student in cache:
        print "Used cache!"
        return cache[student]
    else:
        house = slow_alg(student)
        cache[student] = house
        return house
```

Tip: Take Advantage!

```
> sorting_hat('Amy Hanlon')
'Ravenclaw'
```

Tip: Take Advantage!

```
> sorting_hat('Amy Hanlon')
'Ravenclaw'
> sorting_hat('Amy Hanlon')
Used cache!
‘Ravenclaw'
```

Tip: Take Advantage!

```
> sorting_hat('Amy Hanlon')
'Ravenclaw'
> sorting_hat('Amy Hanlon')
Used cache!
‘Ravenclaw'
> sorting_hat.func_defaults
({'Amy Hanlon': 'Ravenclaw'},)
```

Scope

Trivia

```
> a = 1
```

Trivia

```
> a = 1
> def foo():
    return a
```

Trivia

```
> a = 1
> def foo():
      return a

> foo()
???
```

Trivia

```
> a = 1
> def foo():
    return a

> foo()
1
```

Why?

Namespaces!

- `locals()`

Why?

Namespaces!

- `locals()`
- `globals()` `# {'a' : 1}`

Why?

Namespaces!

- `locals()`
- `globals()` `# {'a' : 1}`
- `__builtins__`

Trivia

```
> a = 1
> def foo():
    return a

> foo()
1
```

Trivia

```
> a = 1
> def foo():
    a += 1
    return a

> foo()
???
```

Trivia

```
> a = 1
> def foo():
    a += 1
    return a

> foo()
UnboundLocalError: local
variable 'a' referenced before
assignment
```

Why?

"When you make an assignment to a variable in a scope, that variable becomes local to that scope."

Why?

```
> a = 1
> def foo():
    # a += 1
    a = a + 1
    return a
```

Tip: Use `global`

```
> a = 1
> def foo():
    global a
    a += 1
    return a
> foo()
2
> a
2
```

Tip: Don't Use `global`

```
> def foo(a):
      a += 1
      return a
> a = 1
> a = foo(a)
2
```

Links

- https://docs.python.org/3/
- http://eli.thegreenplace.net/
- http://akaptur.github.io/blog/2013/10/29/a-python-puzzle/
- http://mathamy.com/python-wats-mutable-default-arguments.html

Thank you!

Questions?