On your way in…

Hand-in:

1. Homework 3s due today
    - 2 piles: SU Boxes < 1700 and SU Boxes >= 1700

Pick-up:

1. POGIL Activity 20: Dictionaries

THIS WEEK'S LAB IS A PARTNERS LAB!

- Your partner must be in your lab section.
- Partner sign-up document:

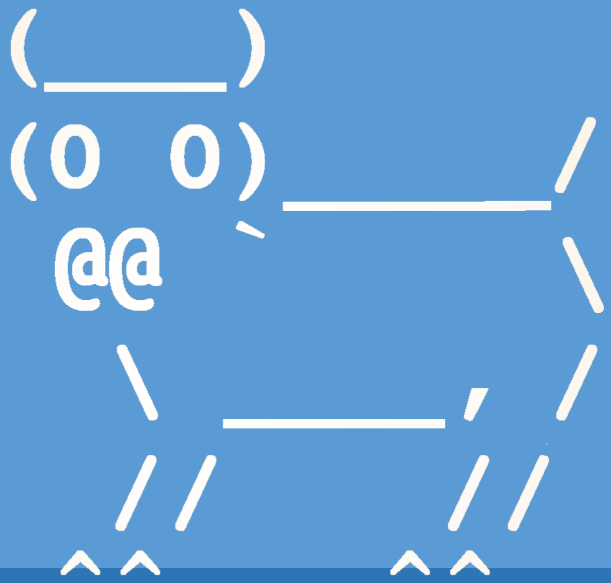
http://www.bit.ly/s20partners

Welcome to CS 134!

Introduction to Computer Science

Iris Howley

-Sets & Dictionaries-



Spring 2020

Midterm Exam is Thursday, March 12

- TPL 203
- 5:45pm-7:45pm OR 8-10pm

- Closed book exam
- Review your homeworks! POGILs! Slides! Labs!

- Next week’s lab will be less intense

- We’ll talk about topic coverage on Wednesday

Some Useful List Functions

```
>>>lst = [1,2,’three’]
>>>lst.append(4.0)
```

Adds one object to end of list

```
   >>>lst        [1,2,’three’,4.0]
>>>lst.extend([5,6])
```

Adds individual elements from sequence to end of list

```
   >>>lst  [1,2,’three’,4.0,5,6]
>>>lst.pop()
```

Removes & returns last element in list

- 6

```
   >>>lst        [1,2,’three’,4.0,5]
>>>l.remove(4.0)
```

Finds & removes given object from list

```
   >>>lst        [1,2,’three’, 5]
```

‘pydoc3 list’ has a lot more!

Some Useful List Functions

```
>>>lst = [1,2,'three',1]
>>>lst.count(1)
```

Counts the number of occurrences of an object in the list

- 2

```
>>>lst.index('three')
    >>>2
```

Returns the index of an object in the list

```
>>>lst.insert(3, 4.0)
```

Inserts an object at a given index: **`insert(index, obj)`**

```
    >>>lst  [1,2,'three',4.0,1]
>>>lst.reverse()
```

Destructively reverses the list

```
    >>>lst  [1,4.0,'three',2,1]
```

'pydoc3 list' has a lot more!

TODAY'S LESSON

Sets

(a mutable data structure that stores unique elements)

Sets

```
>>>s =
 {5,5,5,7,7,7,2,2,2,2,2,2,2,2,2}
>>>s
  {2, 5, 7}
>>>list(s)
  [2, 5, 7]
>>>tuple(s)
  (2, 5, 7)
```

What happened?

No repeats!

Order isn't preserved!

Using Sets

`>>>s = {4,3,3,3,9,1,1}`

`>>>s`

- `{9, 3, 4, 1}`

`>>>s[0]`

- `TypeError: 'set' object does not support indexing`

`>>>s.add(2019)`

`>>>s`

- `{1, 3, 4, 2019, 9}`

‘pydoc3 set’

Immutable Sets?

- Sets are mutable, so if we want an immutable version:

- `s = {5,5,5,7,7,7,2,2,2,2,2,2,2,2,2}`
- `fs = frozenset(s)`

Counting Vocabularies

```
with open(filename) as f:
    wordlist = []
    for line in f:
        line = line.strip()
        wordlist.extend(line.split())
    vocab = set(wordlist)
    len(vocab)
```

s = “hello there folks!”

s.split(“ “) → [‘hello’,’there’,’folks’]

Counting Vocabularies

- Just because you can’t `myset[0]`, doesn’t mean you can’t iterate over elements in a set!

- `for item in vocab:`
  - `if item in [‘wizard’, ‘harry’]`
    - `print(item)`

Set Functions

- `s = {1,2,3,4}`
- `s2 = {1,2}`
- `s2.issubset(s)` → `True`
- `s.issubset(s2)` → `False`
- `s.issuperset(s2)` → `True`
- `s2.add(99)`
- `s2.issubset(s)` → `False`

- `s.union(s2)`
  - `{1,2,3,4,99}`
- `s.intersection(s2)`
  - `{1,2}`
- `s.difference(s2)`
  - `{3,4}`
- `s2.difference(s)`
  - `{99}`

‘pydoc3 set’

TODAY'S LESSON

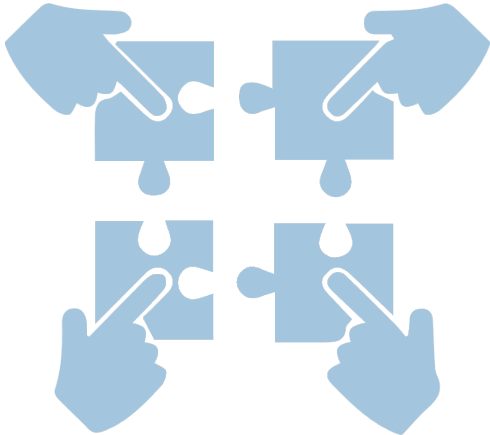
Dictionaries

(a data structure with convenient indexing, no iteration needed!)

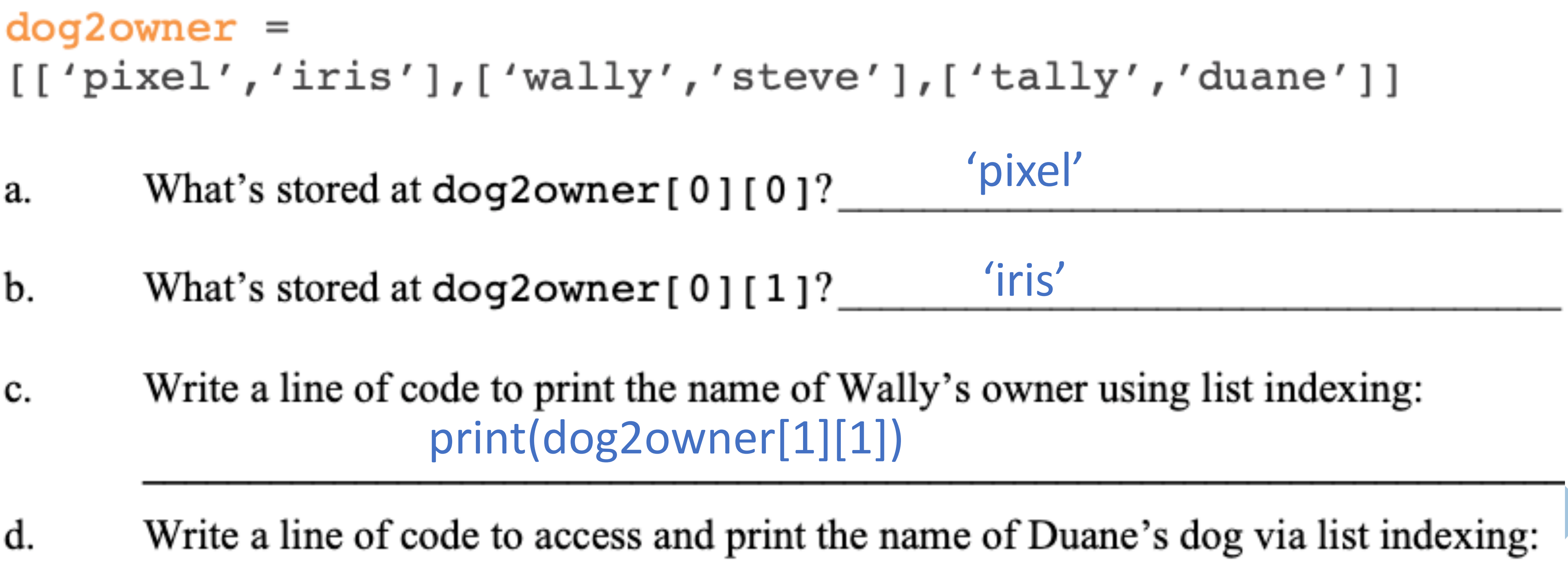
POGIL Activity 20- Dictionaries

- Stores data that can be accessed via meaningful indices

- Look at Python Activity 20, Question 1-5
- Find a partner and talk through the questions together



POGIL – Activity 20: Question 1



```
dog2owner =
[['pixel','iris'],['wally','steve'],['tally','duane']]
```

a. What's stored at `dog2owner[0][0]`? ‘pixel’

b. What's stored at `dog2owner[0][1]`? ‘iris’

c. Write a line of code to print the name of Wally's owner using list indexing:

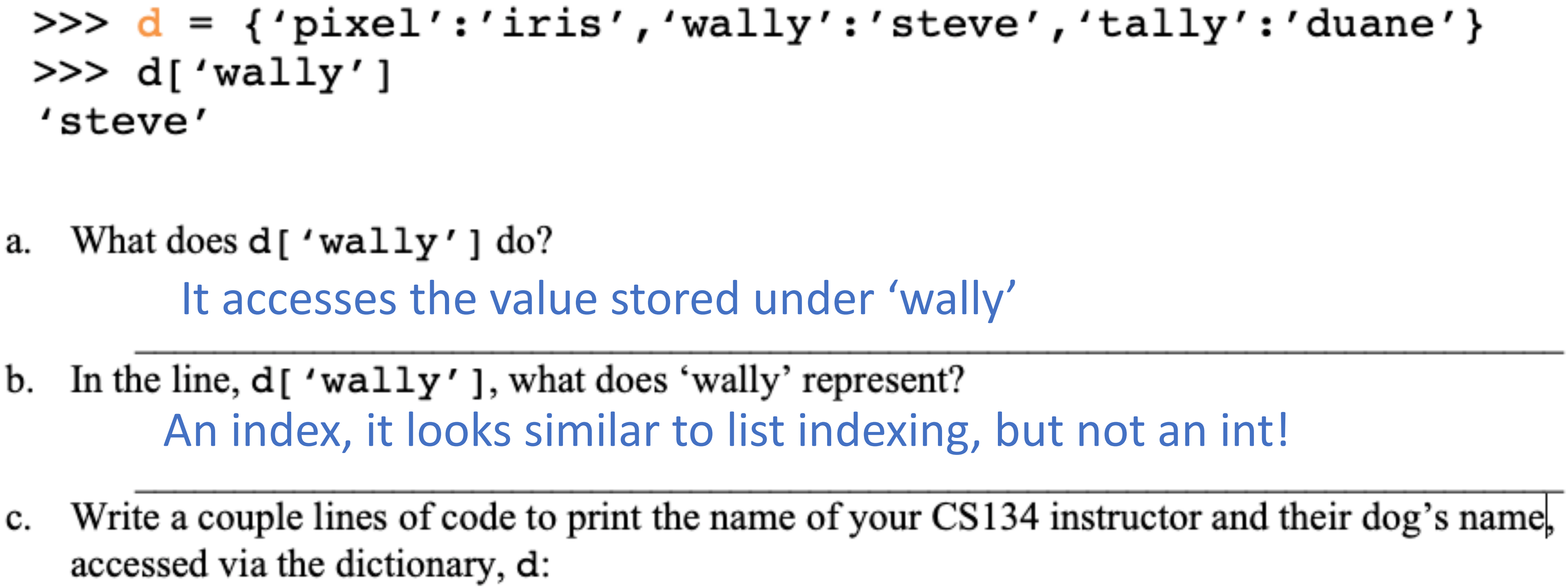
```
print(dog2owner[1][1])
```

d. Write a line of code to access and print the name of Duane's dog via list indexing:

```
print(dog2owner[2][0])
```

THERE'S A MUCH EASIER WAY TO STORE/ACCESS THIS TYPE OF DATA!

POGIL – Activity 20: Question 2



```
>>> d = {‘pixel’:’iris’,‘wally’:’steve’,‘tally’:’duane’}
>>> d[‘wally’]
‘steve’
```

a. What does `d[‘wally’]` do?

It accesses the value stored under ‘wally’

b. In the line, `d[‘wally’]`, what does ‘wally’ represent?

An index, it looks similar to list indexing, but not an int!

c. Write a couple lines of code to print the name of your CS134 instructor and their dog’s name, accessed via the dictionary, `d`:

```
key = ‘pixel’
print(“{}’s dog is {}”.format(d[key], key))
```

Dictionary Syntax

Make an empty dictionary w `{}` or `dict()`

- `aDiction = {}`

Curly brackets (with colon indices) mean dictionary

- `aDiction = {‘key’: ‘value’, 2: [var1, var2]}`

Maps a key on the left-hand side to a value on the right-hand side

Can use key values as index

- `print(aDiction[‘key’])`
  - `‘value’`

Dictionary Keys

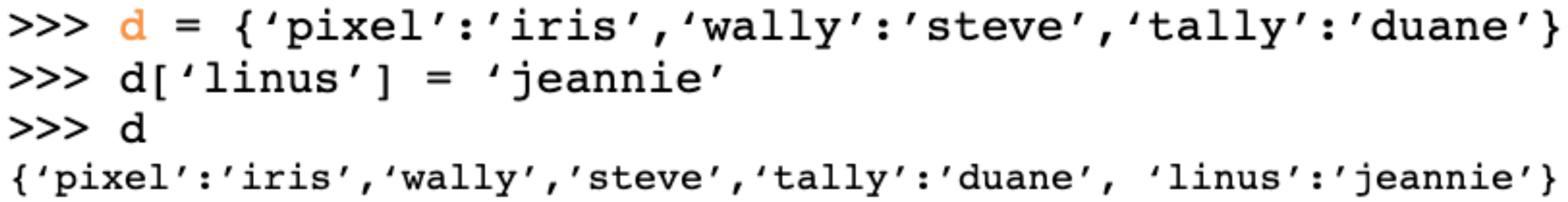
- `d[[‘bill l’,’bill j’] = ‘williams college’`
  - ERROR
- `d[(‘bill l’,’bill j’)] = ‘williams college’`
  - `d`
    - `{('bill l', 'bill j'): 'williams college’}`

What’s the difference?

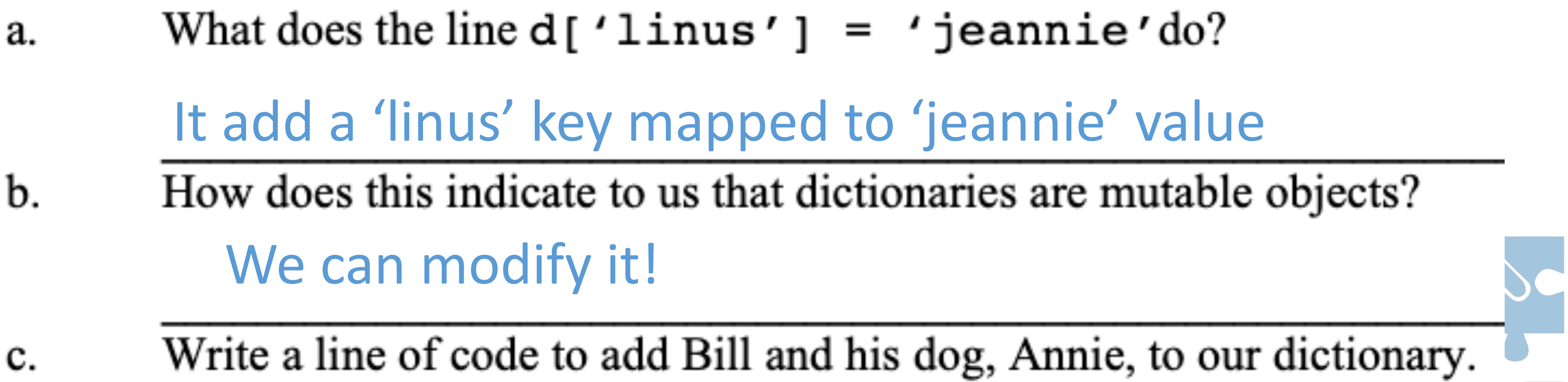
Dictionary keys must be immutable types

`int, float, string, bool, tuple, frozenset`

POGIL – Activity 20: Question 3



```
>>> d = {‘pixel’:’iris’,‘wally’:’steve’,‘tally’:’duane’}
>>> d[‘linus’] = ‘jeannie’
>>> d
{‘pixel’:’iris’,‘wally’,’steve’,‘tally’:’duane’, ‘linus’:’jeannie’}
```



a. What does the line `d[‘linus’] = ‘jeannie’` do?

It add a ‘linus’ key mapped to ‘jeannie’ value

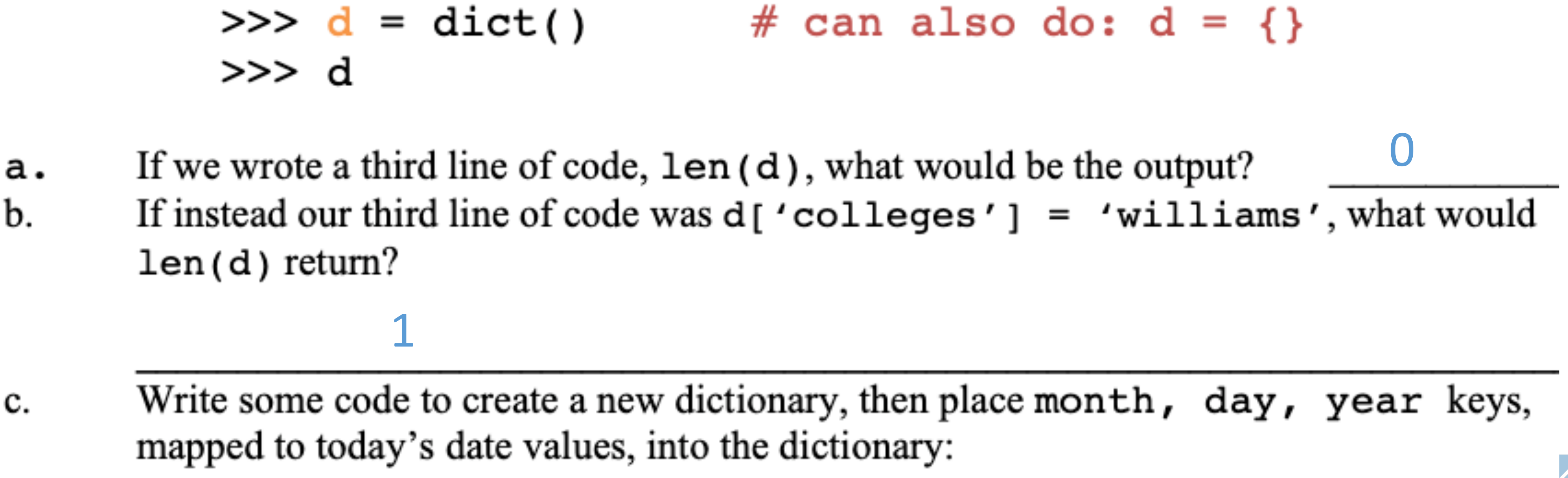
b. How does this indicate to us that dictionaries are mutable objects?

We can modify it!

c. Write a line of code to add Bill and his dog, Annie, to our dictionary.

d[‘annie’] = ‘bill’

POGIL – Activity 20: Question 4



```
>>> d = dict()          # can also do: d = {}
>>> d
```

a. If we wrote a third line of code, `len(d)`, what would be the output? 0

b. If instead our third line of code was `d['colleges'] = 'williams'`, what would `len(d)` return?

1

c. Write some code to create a new dictionary, then place `month, day, year` keys, mapped to today's date values, into the dictionary:

```
d = dict()
d['m'] = 3
d['d'] = 2
d['y'] = 2020
```

POGIL – Activity 20: Question 5

```
>>> d = {}                  # can also do: d = dict()
>>> d[‘colleges’] = ‘williams’
>>> d[‘colleges’] = ‘amherst’
```

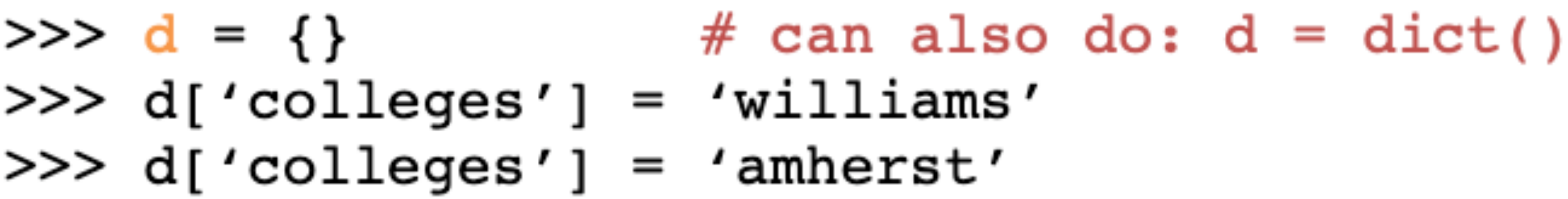
a. If we wrote a fourth line of code, `print(d)`, what might be the output?

{‘colleges’:’amherst’}

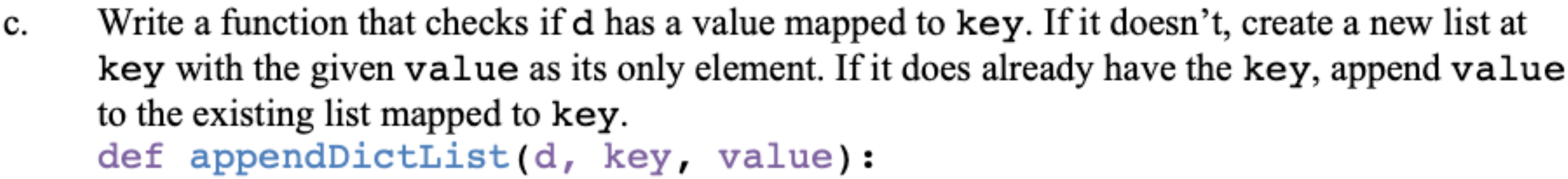
b. At the end of this code execution, `d` only has: `{‘colleges’: ’amherst’}` Why might this be?

Dictionaries can only hold one key, any repeats override!

POGIL – Activity 20: Question 5



```
>>> d = {}                    # can also do: d = dict()
>>> d[‘colleges’] = ‘williams’
>>> d[‘colleges’] = ‘amherst’
```



c. Write a function that checks if `d` has a value mapped to `key`. If it doesn’t, create a new list at `key` with the given `value` as its only element. If it does already have the `key`, append `value` to the existing list mapped to `key`.

```
def appendDictList(d, key, value):
```

Dictionary `.get(object, missingReturnObject)` is convenient here!

Dictionary **.get()**

Keys

- aDiction = {‘key’: ‘value’, 2: ‘hello’}

.Get function for dictionaries

Key for which you want a value

- aDiction.get(2, -1)

  Value to return if the key isn’t there

  - ‘hello’

    Key exists, so return the value

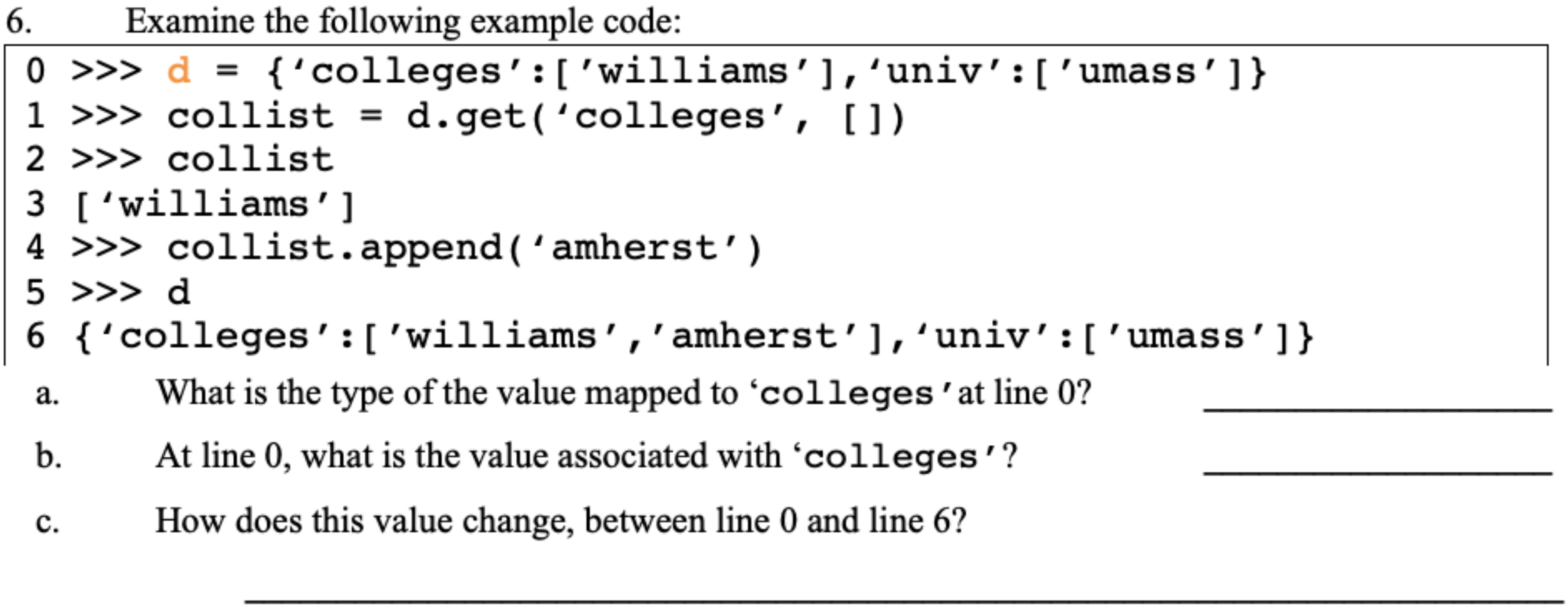
- aDiction.get(3333, -1)
  - -1

    Key doesn’t exist, so return the value we opted for

SUPER handy when the values are lists/sequences!

Make missing return value []

POGIL – Activity 20: Question 6



6. Examine the following example code:

```
0 >>> d = {‘colleges’:[’williams’],‘univ’:[’umass’]}
1 >>> collist = d.get(‘colleges’, [])
2 >>> collist
3 [‘williams’]
4 >>> collist.append(‘amherst’)
5 >>> d
6 {‘colleges’:[’williams’,’amherst’],‘univ’:[’umass’]}
```

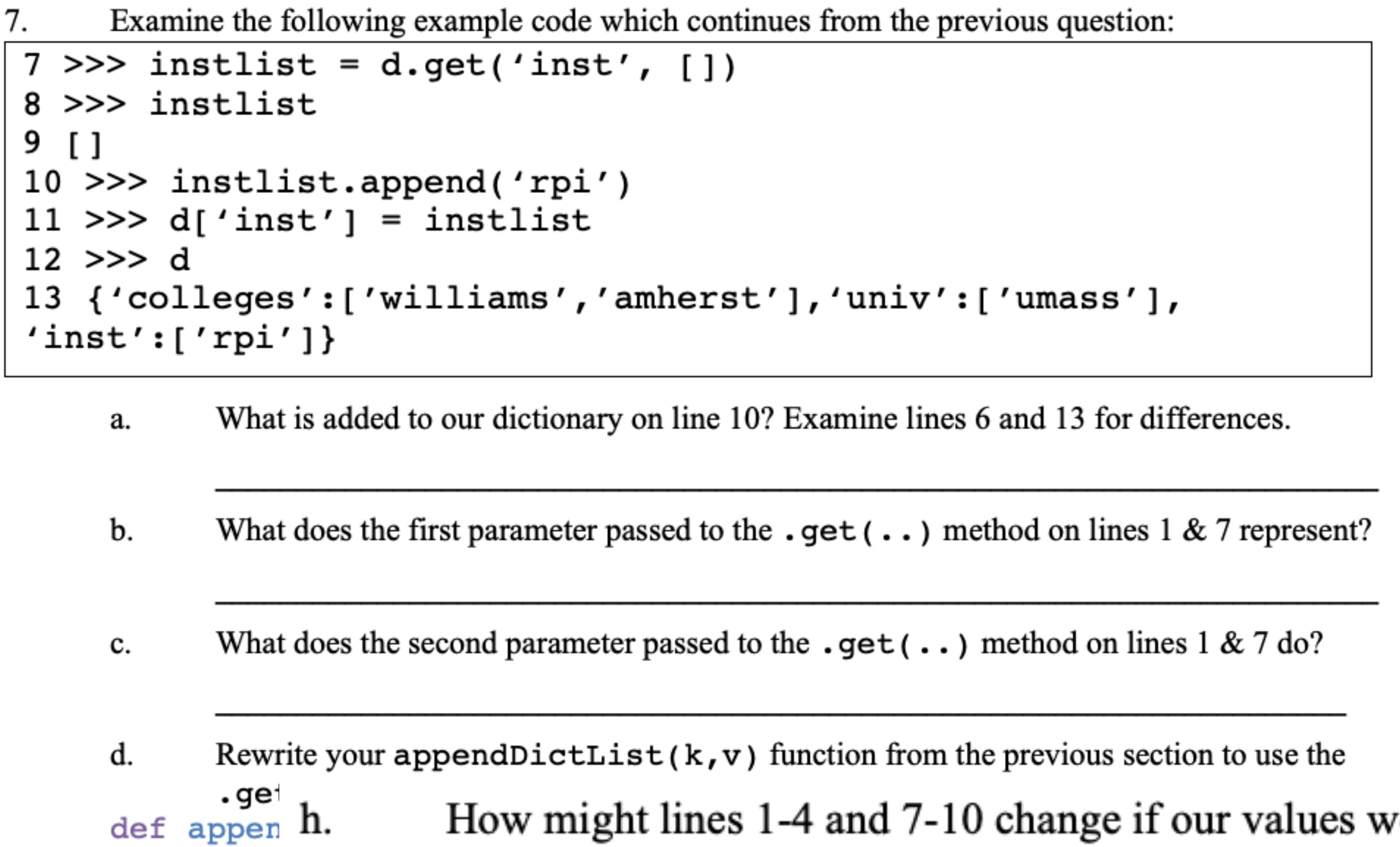
a. What is the type of the value mapped to ‘`colleges’` at line 0? ____________________

b. At line 0, what is the value associated with ‘`colleges’`? ____________________

c. How does this value change, between line 0 and line 6?

____

POGIL – Activity 20: Question7



7. Examine the following example code which continues from the previous question:

```
7 >>> instlist = d.get(‘inst’, [])
8 >>> instlist
9 []
10 >>> instlist.append(‘rpi’)
11 >>> d[‘inst’] = instlist
12 >>> d
13 {‘colleges’:[’williams’,’amherst’],‘univ’:[’umass’],
‘inst’:[’rpi’]}
```

a. What is added to our dictionary on line 10? Examine lines 6 and 13 for differences.

____

b. What does the first parameter passed to the `.get(..)` method on lines 1 & 7 represent?

____

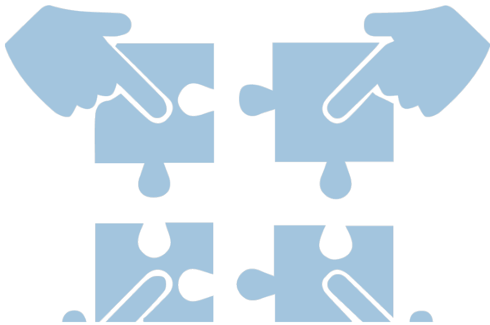
c. What does the second parameter passed to the `.get(..)` method on lines 1 & 7 do?

____

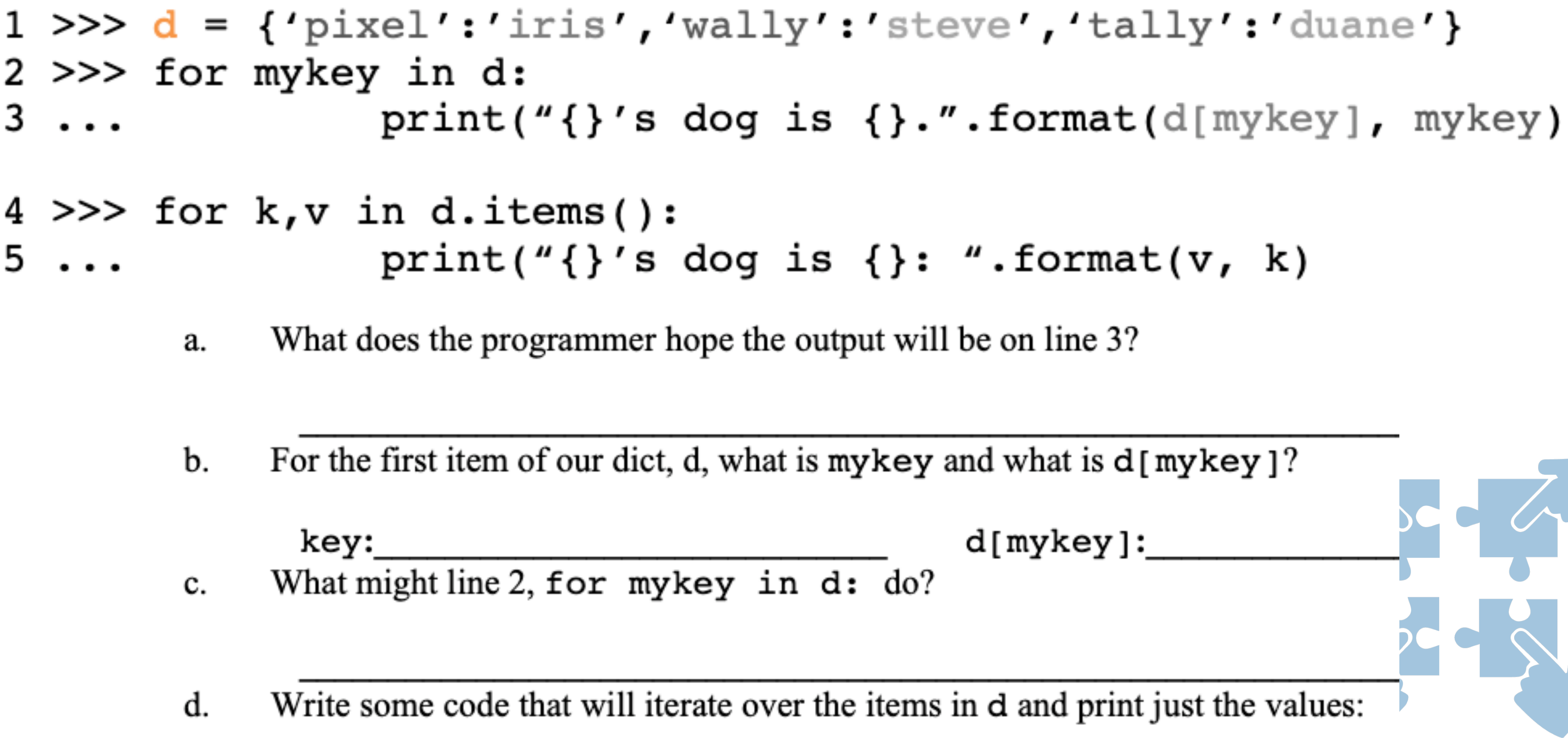
d. Rewrite your `appendDictList(k,v)` function from the previous section to use the `.ge`

```
def appen
```

h. How might lines 1-4 and 7-10 change if our values were strings instead of lists?



POGIL – Activity 20: Question 8



```
1 >>> d = {'pixel':'iris','wally':'steve','tally':'duane'}
2 >>> for mykey in d:
3 ...         print("{}'s dog is {}.".format(d[mykey], mykey)

4 >>> for k,v in d.items():
5 ...         print("{}'s dog is {}: ".format(v, k)
```

a. What does the programmer hope the output will be on line 3?

____

b. For the first item of our dict, d, what is `mykey` and what is `d[mykey]`?

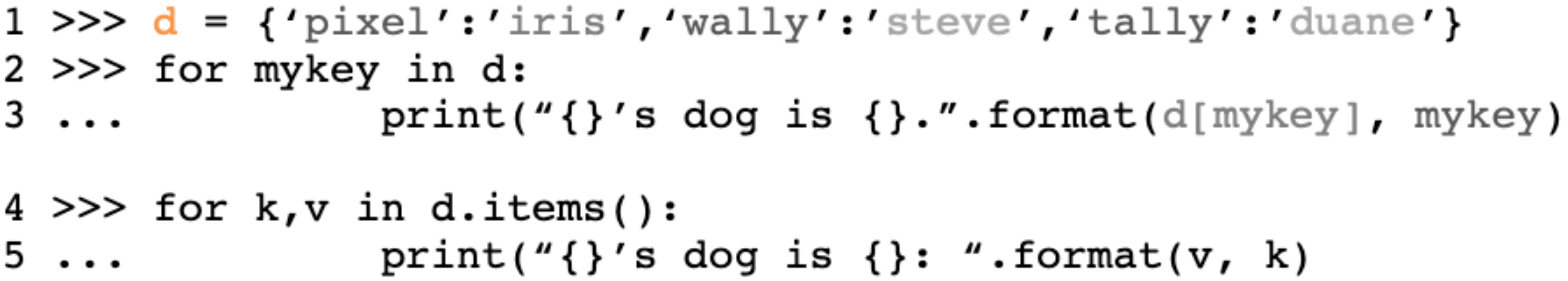
`key:`____________________________ `d[mykey]:`______________

c. What might line 2, `for mykey in d:` do?

____

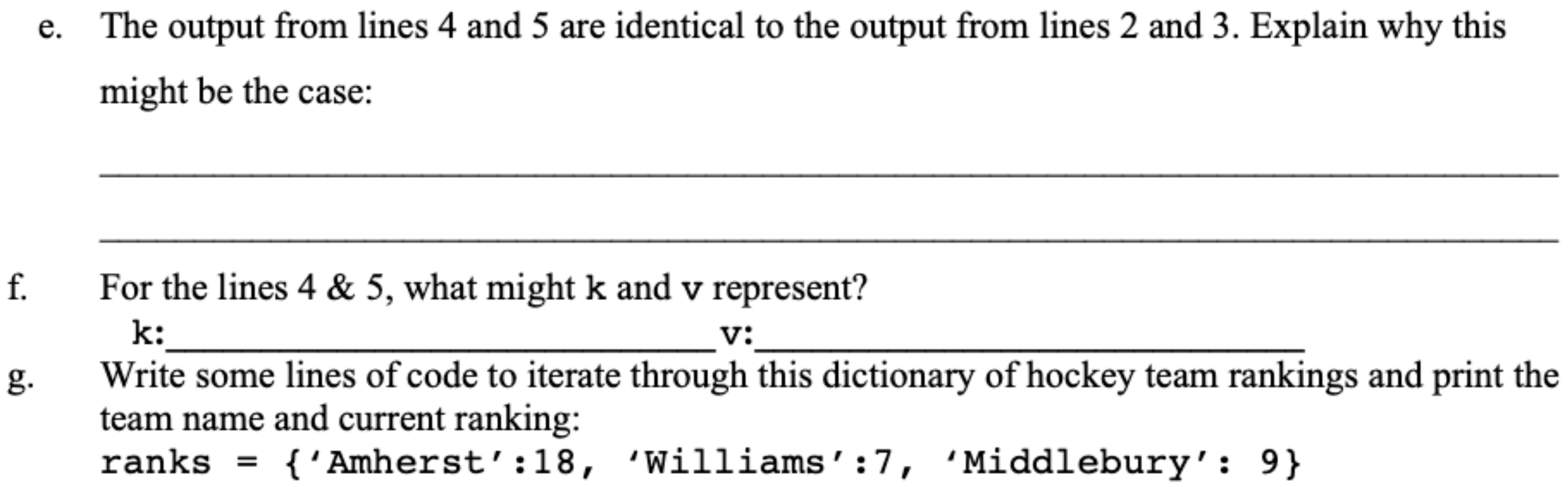
d. Write some code that will iterate over the items in `d` and print just the values:

POGIL – Activity 20: Question 8



```
1 >>> d = {‘pixel’:’iris’,‘wally’:’steve’,‘tally’:’duane’}
2 >>> for mykey in d:
3 ...           print(“{}’s dog is {}.”.format(d[mykey], mykey)

4 >>> for k,v in d.items():
5 ...           print(“{}’s dog is {}: “.format(v, k)
```



e. The output from lines 4 and 5 are identical to the output from lines 2 and 3. Explain why this might be the case:

____

____

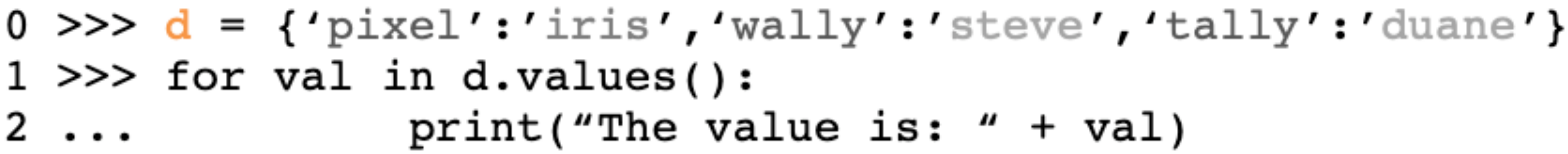
f. For the lines 4 & 5, what might k and v represent?

k:______________________________ v:______________________________

g. Write some lines of code to iterate through this dictionary of hockey team rankings and print the team name and current ranking:

```
ranks = {‘Amherst’:18, ‘Williams’:7, ‘Middlebury’: 9}
```

POGIL – Activity 20: Question 9



```
>>> d = {‘pixel’:’iris’,‘wally’:’steve’,‘tally’:’duane’}
>>> for val in d.values():
...             print(“The value is: “ + val)
```

a. What might the line `for val in d.values:` do?

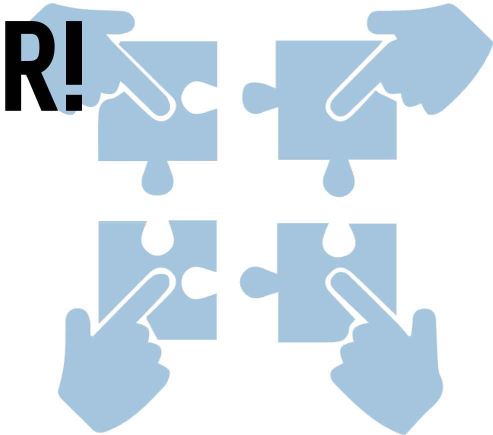
____

b. What would you guess the output of this code to look like?

YOU SHOULD COMPLETE THE REST OF ALL POGILS OUTSIDE OF CLASS.

BEST DONE WITH A PARTNER OR STUDY GROUP.

CHECK YOUR ANSWERS ON A COMPUTER!



Lists of Lists

- `dog2owner = [[‘pixel’,‘iris’],[‘wally’,’steve’],[‘tally’,’duane’]]`

- What index is the name of Tally’s owner at within dog2owner?
    - Just the owner’s name!

Take a minute to discuss with a partner

Lists of Lists

- `dog2owner = [['pixel','iris'],['wally','steve'],['tally','duane']]`
- What index is the name of Tally's owner at within dog2owner?

1. What is the index of the element of dog2owner that we want?
   - dog2owner[0] → ['pixel', 'iris']
   - dog2owner[1] → ['wally', 'steve']
   - dog2owner[2] → ['tally', 'duane']
2. What is the index of the element within that element, that we want?
   - ['tally', 'duane'][0] → 'tally'
   - ['tally', 'duane'][1] → 'duane'

`dog2owner[2][1]`

- l = [’pixel', ’wally', ’tally’]
- l[1]
  - ‘wally’

- d = {'pix':'iris','wally':'steve','tally':'duane'}

- d[‘tally’]
  - ‘duane’

Dictionaries

Key Value

- `d = {'pix':'iris','wally':'steve','tally':'duane'}`
- `d[‘tally’]`
  - `‘duane’`
- `d[‘pix’]`
  - `‘iris’`
- `d[‘wally’]`
  - `‘steve’`

Mapping from Key to Value

Iterating Over Dictionaries

- `d = {'pix':'iris','wally':'steve','tally':'duane'}`
- `for key in d:`
  - `print(“{}’s dog is {}“.format(d[key],key))`

  - When key is ‘pix’:
    - `iris’s dog is pix` d[key] is ‘iris’
  - When key is ‘wally’:
    - `steve’s dog is wally` d[key] is ‘steve’
  - When key is ‘tally’:
    - `duane’s dog is tally` d[key] is ‘duane’

Dictionary Keys

- `d = {1:’hello’, 2:2019}`
  - Keys can be other types, so can values

- `d[‘good’] = [‘bye‘] * 3`
  - We can add values mapped to a specified key
  - `{1: 'hello', 2: 2019, 'good': ['bye’,’bye’,’bye’]}`

Dictionary Keys

- d = {‘bill’: ‘Dartmouth’}
- d[‘bill’] = ‘Stony Brook U’
- d
  - {‘bill’: ‘Stony Brook U’}
  - Only one key with same value! Overwrites!

Prof. Bill Lenhart

Prof. Bill Jannen

- d[‘bill’] = [‘Dartmouth’, ‘Stony Brook U’]
  - ...But lists can also be dictionary values

Dictionary Keys

- d[[‘bill l’,’bill j’]] = ‘williams college’
  - ERROR
- d[(‘bill l’,’bill j’)] = ‘williams college’
  - d
    - {('bill l', 'bill j'): 'williams college’}

What’s the difference?

Dictionary keys must be immutable types

int, float, string, bool, tuple, frozenset

Detecting if Something in a Dictionary

- d ={‘dogs’:5, ’cats’:1}
- ‘cats’ in d
  - True
- 5 in d → 5 in d.values()
  - False

  True

- l = [’pix', ’wally', ’tally’]
- if ‘wally’ in l:
  - print(“Found Wally!”)

Dictionaries

- `d = dict()`
- `d = {4:101, 2:760, 9: 422}`
- `list(d)`
  - `[4, 2, 9]`
- `list(d.values())`
  - `[101, 760, 422]`
- `list(d.items())`
  - `[(4,101), (2,760), (9,422)]`

Remember ‘pydoc3 dict’ for more functions!

QUESTIONS?

Leftover Slides

Sets & Frozensets

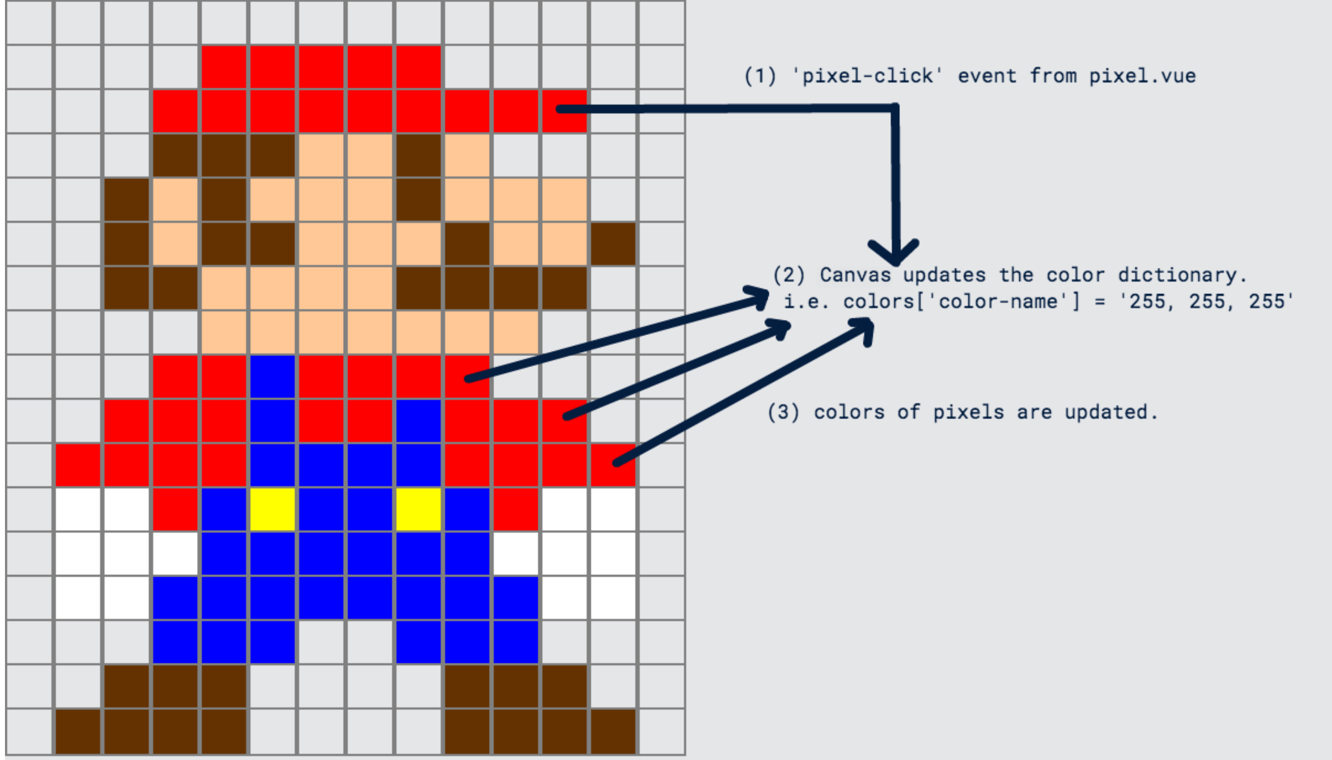
- `s = {4,3,3,3,9,1,1}`
- `s`
  - `{9, 3, 4, 1}`
- `s[0]`
  - `TypeError: 'set' object does not support indexing`
- `s.add(2019)`
- `s`
  - `{1, 3, 4, 2019, 9}`

- `fs = frozenset(s)`
- `fs`
  - `frozenset({1,3,4,2019,9})`
- `fs[0]`
  - `TypeError: 'frozenset' object does not support indexing`
- `fs.add(2019)`
  - `AttributeError: 'frozenset' object has no attribute 'add'`

‘pydoc3 set’

Why Lists of Lists?

Images

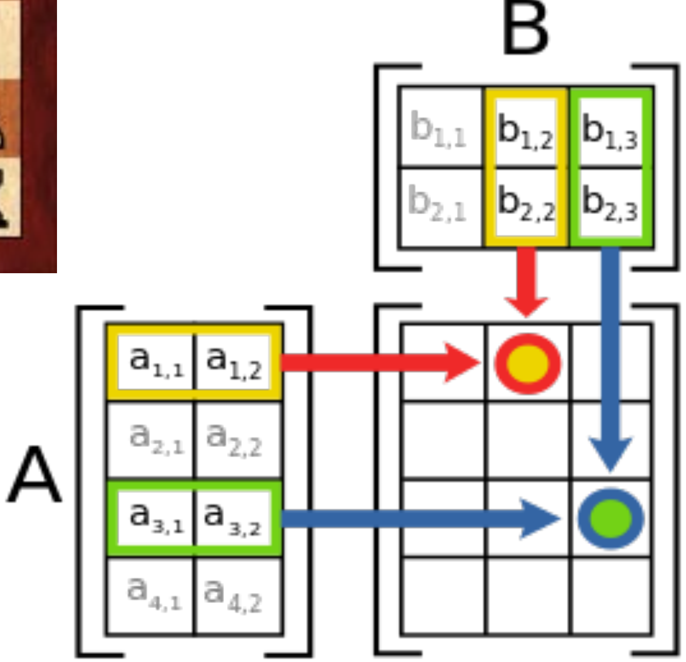


(1) 'pixel-click' event from pixel.vue

(2) Canvas updates the color dictionary.
i.e. colors['color-name'] = '255, 255, 255'

(3) colors of pixels are updated.

Games



Mathematics

HASHING

Finding dictionary values quickly

Dictionary Keys

- d[[‘bill l’,’bill j’]] = ‘williams college’
    - ERROR
- d[(‘bill l’,’bill j’)] = ‘williams college’
    - d
        - {('bill l', 'bill j'): 'williams college’}

What’s the difference?

Dictionary keys must be immutable types

int, float, string, bool, tuple, frozenset

Dictionary Keys

Dictionary keys must be immutable types

int, float, string, bool, tuple, frozenset

Why?

Mutable Types as Dictionary Keys

- Lists are mutable
- When you append() to a list, it changes that list object
- If you used a list object as a key in a dictionary, you wouldn't be able to find it again, after it's been changed

```
mylist = [‘a’, ’b’]
mydict = dict()
mydict[mylist] = ‘throws an error’
mylist.append(‘c’)
print(mydict[mylist])
# Now mylist is no longer findable in the dict!
```

We're going to see why!

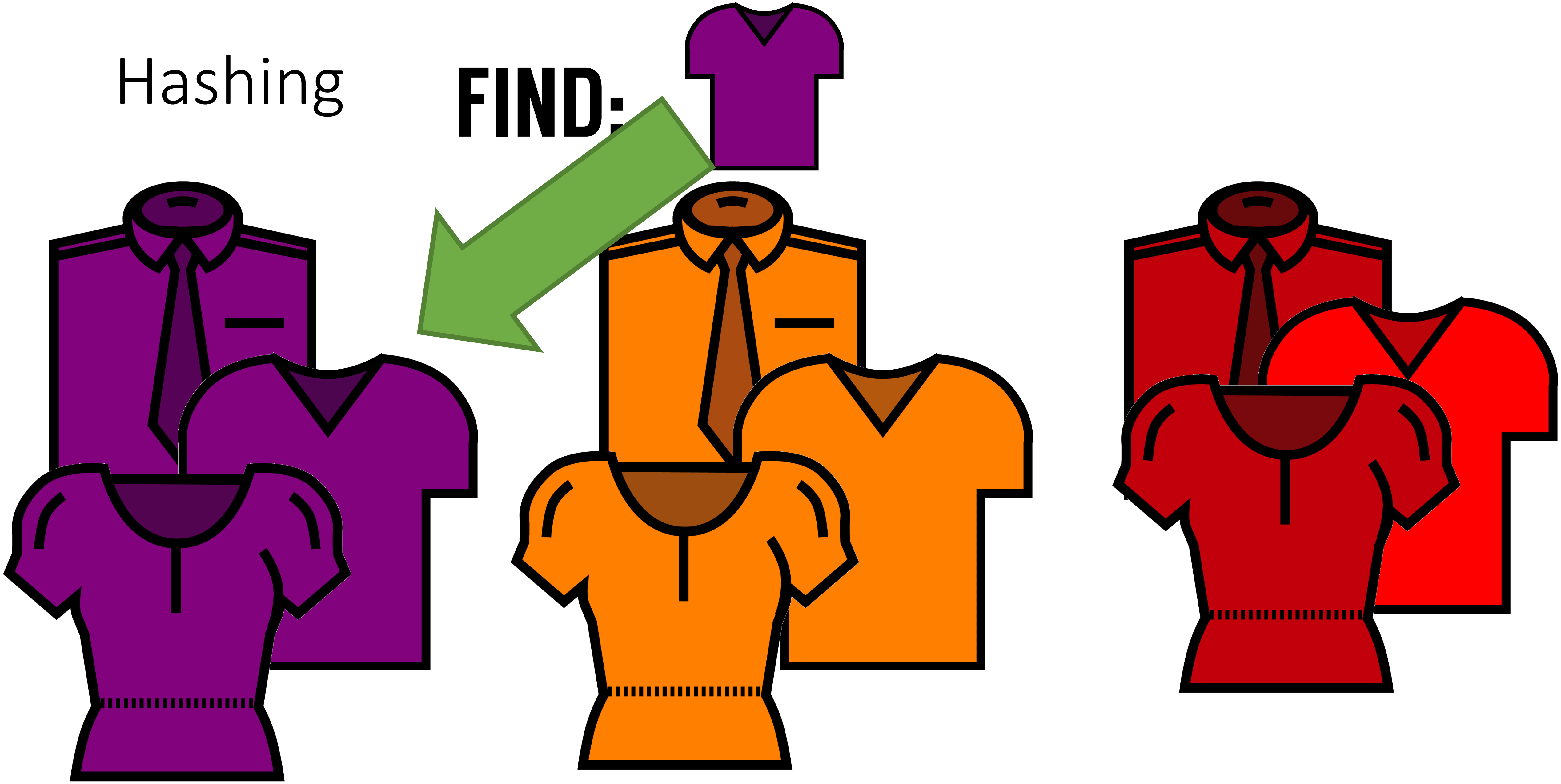
Dictionary Keys

- Dictionaries index their items by a hash
- A hash is an fixed sized integer that identifies a particular value.
- Each value needs to have its own hash
    - For the same value you will get the same hash even if it's not the same object.

Why not just index items based on their value?

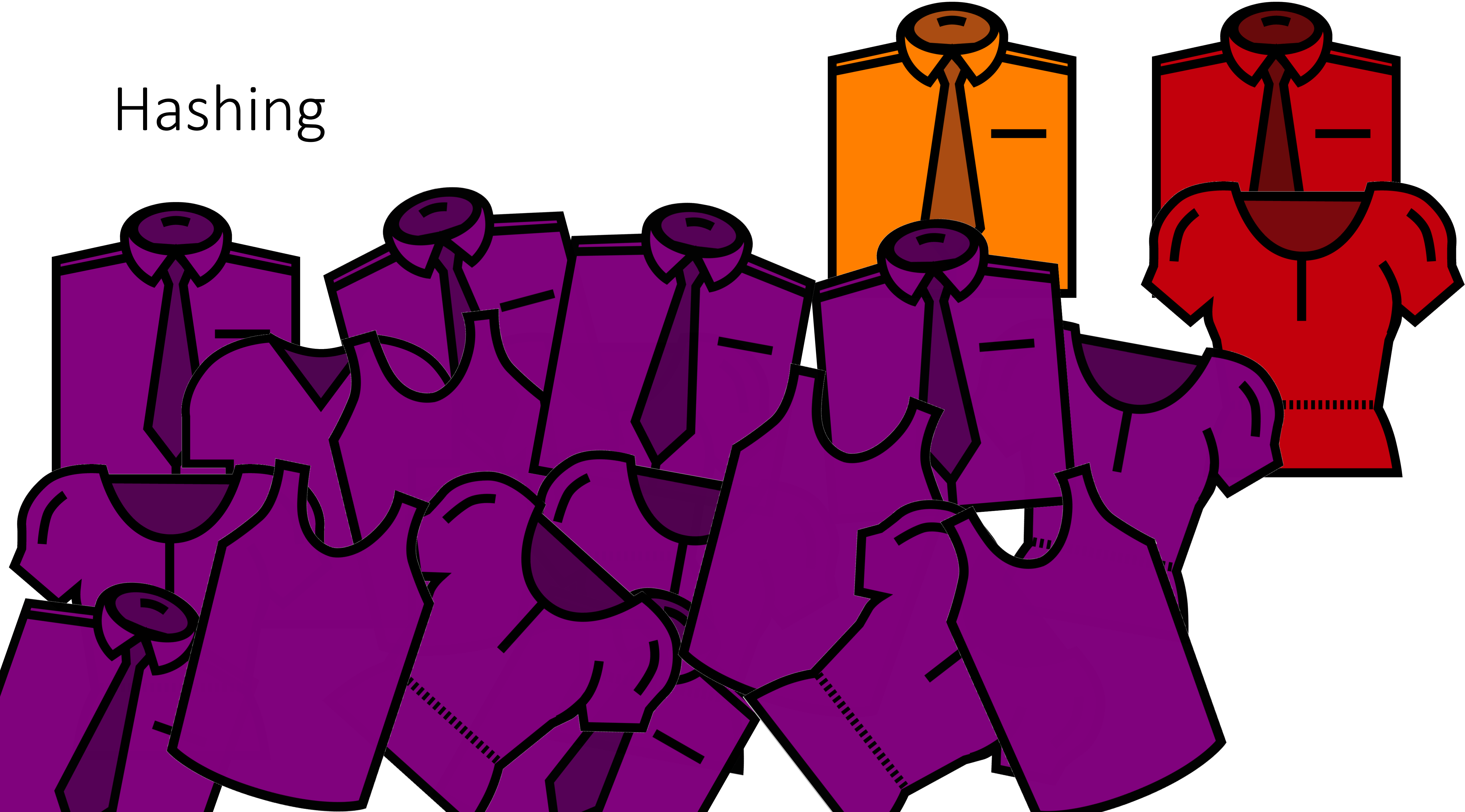
Hashing

Hashing



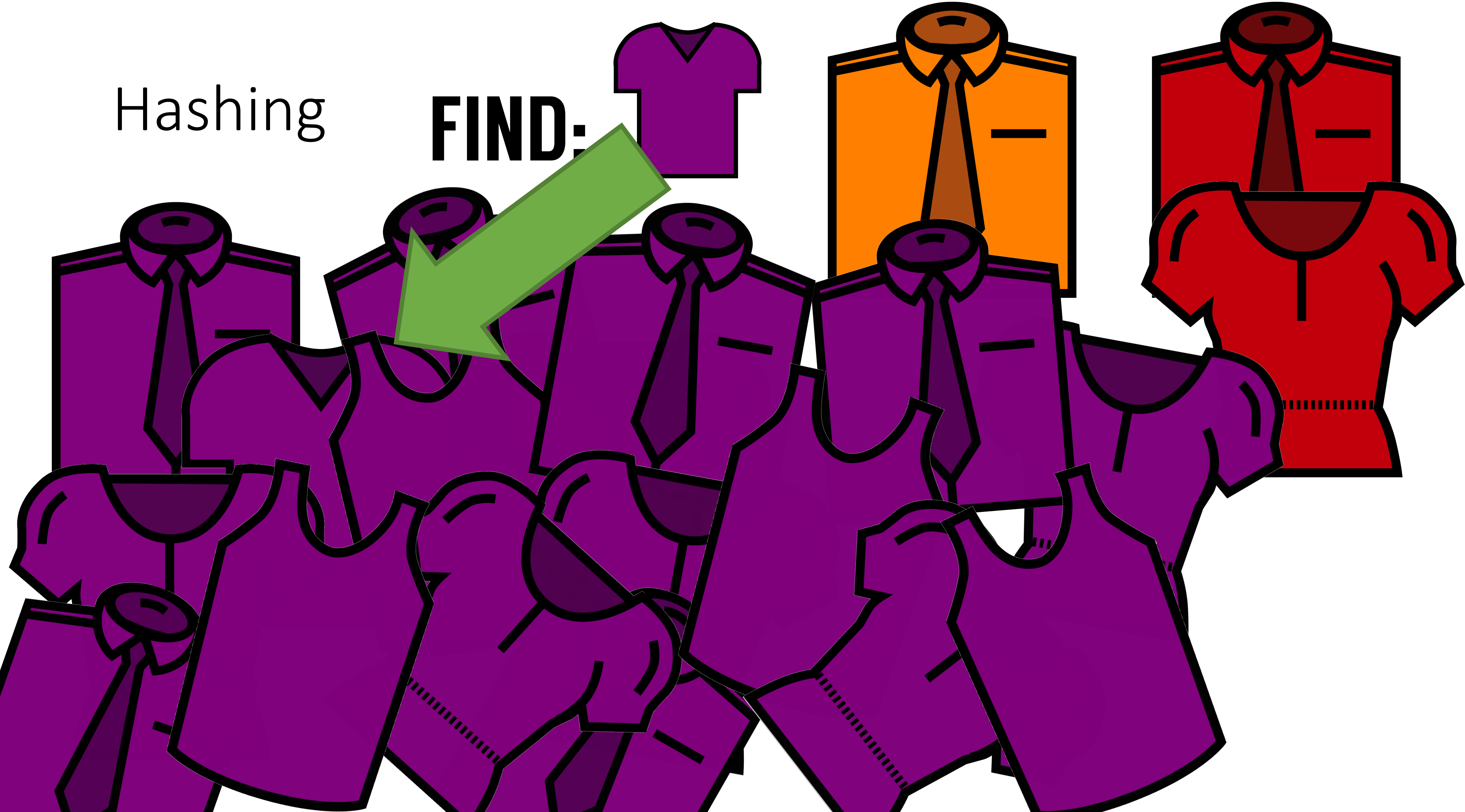
FIND:

Hashing



Hashing

FIND:



Hashing

FIND:

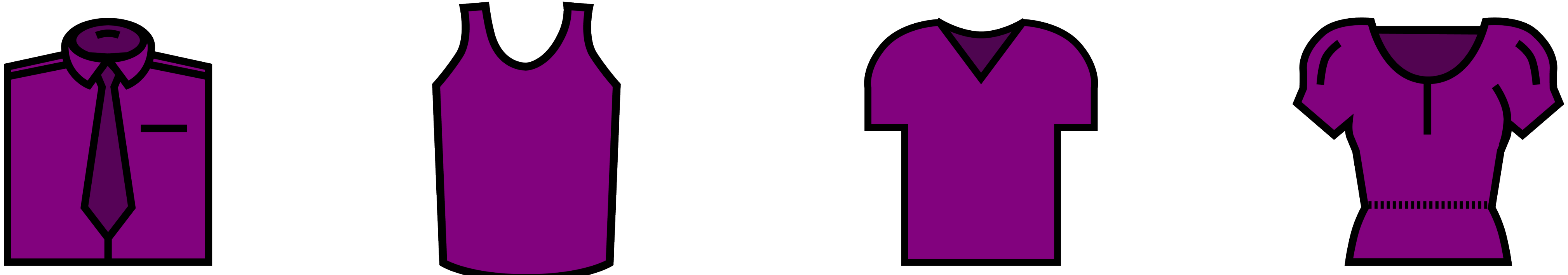
Hashing

Why not just index items based on their value?

- We could organize all words in memory by the letter they start with…

- But words that start with ‘A’ could be numerous
- Compared to words that start with ‘Z’
  - …Sort of like arranging clothes by color

- Hashing is a different way of mapping items to make them easier to find



Hashing

- Other concerns
  - Bad hashing function for your data, resulting in clustering
  - Running out of space in the pile you’ve assigned
  - Placing shirts in the wrong pile

- Stored in the order that makes it easiest to look them up

hash(o) → o.__hash__()

- `s = “hello world”`
- `t = s + “!”`
- `hash(s)` → 4960501519247167238
- `hash(s)` → 4960501519247167238
- `hash(t)` → -8774050965770600213
- `hash(t[:-1])` → 4960501519247167238

If the 2 strings are the same, they’ll get the same hash

If the 2 strings are different, they *might* get a different hash.

hash(o) → o.__hash__()

Some hash codes are expensive (million-long tuple)

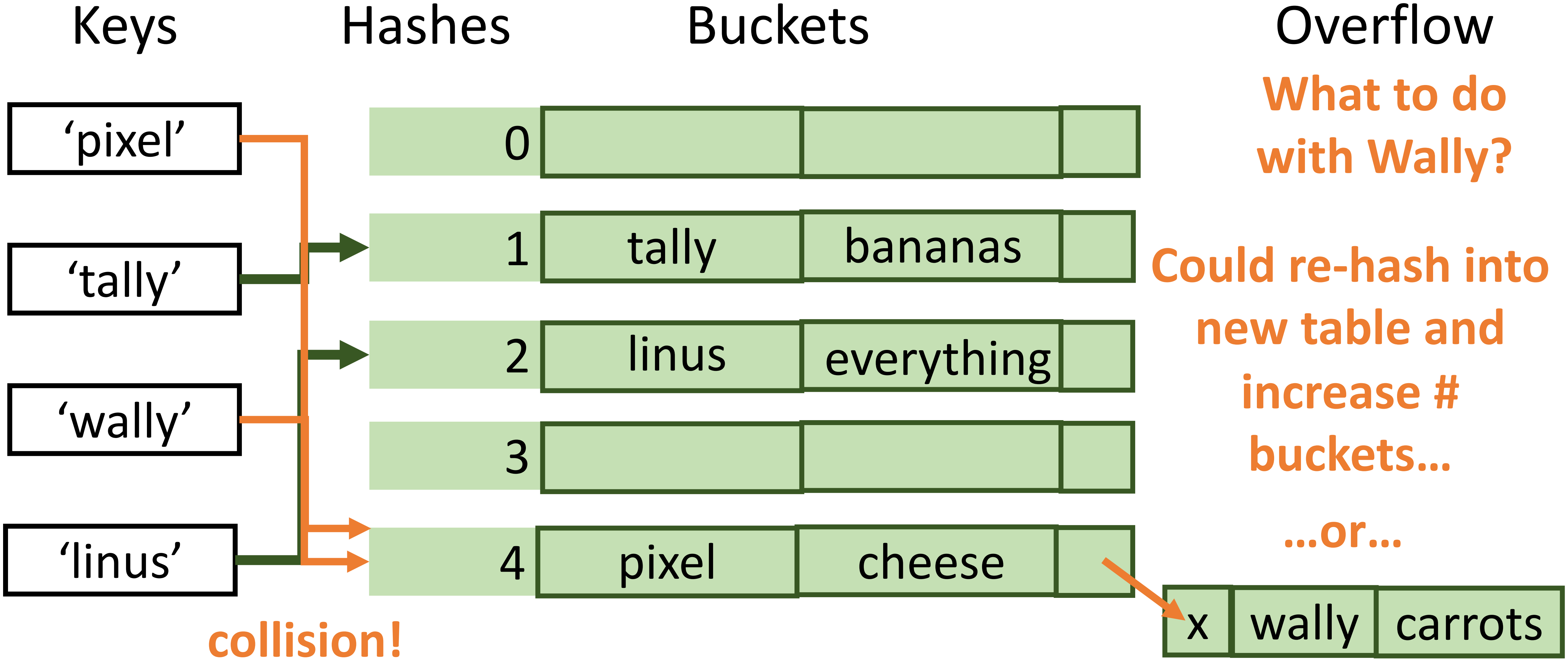
- `hash(1) → 1`
- `hash(2) → 2`
- `hash(1000000000000000000) → 1000000000000000000`
- `hash(10000000000000000000) → 776627963145224196`

At some length, it starts treating the numbers like a string

If the hash codes are the same, the values might be the same

Hash Tables

How to access mydict['wally']?



What to do with Wally?

Could re-hash into new table and increase # buckets…

…or…

Immutable Objects

- Have no way to set/change the attributes, without creating a new object
  - Like `int, string`, etc.
  - Like the `Color` class from this week's lab!
  - `__slots__ = []`
- Can be used in `sets`
  - i.e., you cannot have a `set` of `lists`
- Can be used as keys for `dictionaries`
  - If the class has a `__hash__()` function defined!

Hashing

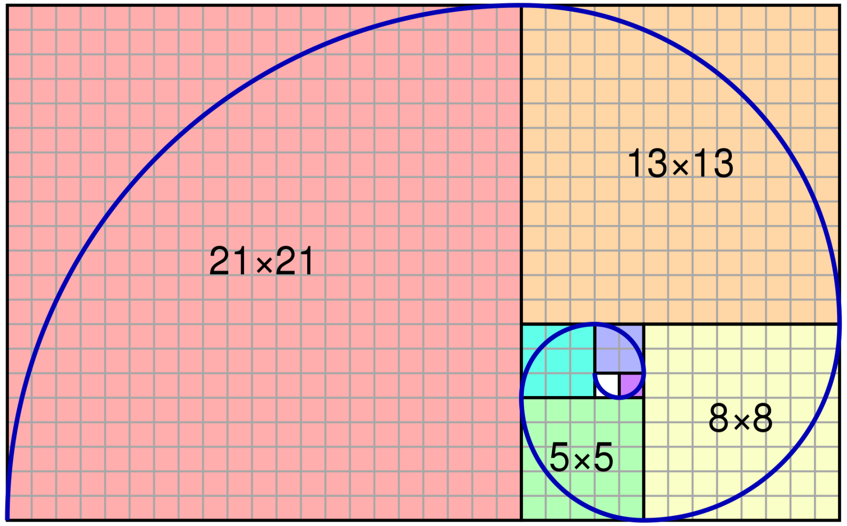
- Don’t know how it’s computed → Abstraction

- There’s many ways to implement a hash function, here’s a description of some of them:
    - https://www.cs.hmc.edu/~geoff/classes/hmc.cs070.200101/homework10/hashfuncs.html

Algorithms

Fibonacci Sequence

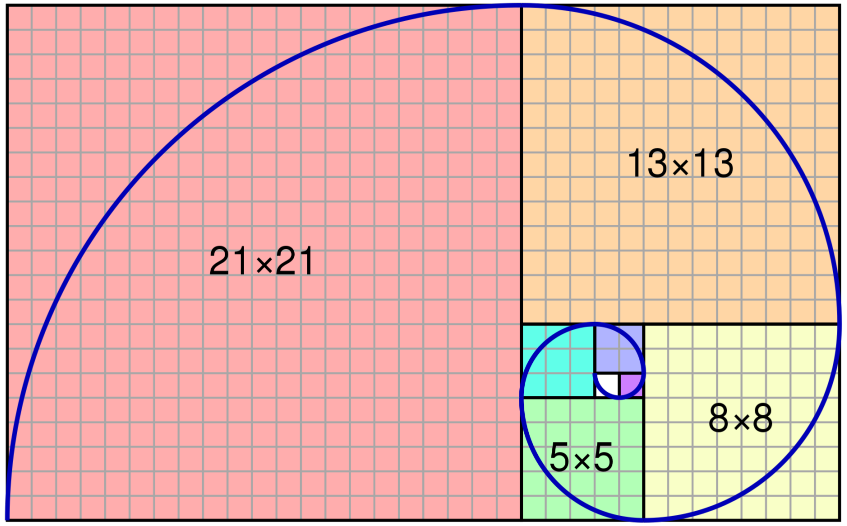
- fibo(0) = 0
- fibo(1) = 1
- fibo(n) = fibo(n-1) + fibo(n-2)
  - fibo(4) = fibo(3) + fibo(2)
    - = fibo(2) + fibo(1) + fibo(1)+fibo(0)
    - = fibo(1)+fibo(0) + 1 + 1 + 0
    - = 1 + 0
    - = 3



Fibonacci Sequence

- fibo(0) = 1 call of fibo()
- fibo(1) = 1 call
- fibo(2) = 3 calls
- fibo(3) = 5 calls
- fibo(4) = 9 calls
- fibo(5) = 15 calls…

- For each increase in n, the number of function calls practically doubles



Speeding Up Fibonacci

(Memoization)

```
global postit
if n in postit:
   answer = postit[n]
else:
   if n < 2:
        answer = n
   else:
        answer = fibo(n-1) + fibo(n-2)
   postit[n] = answer
return answer
```