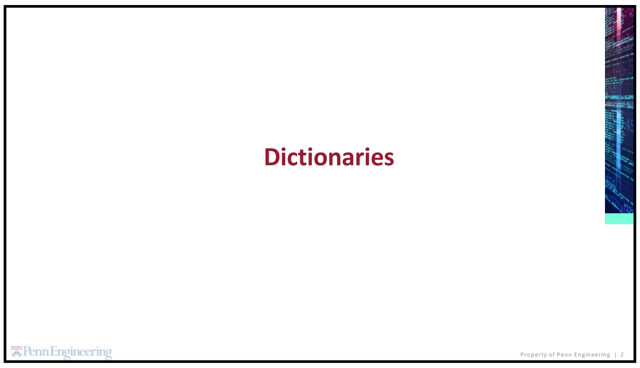
Dictionaries

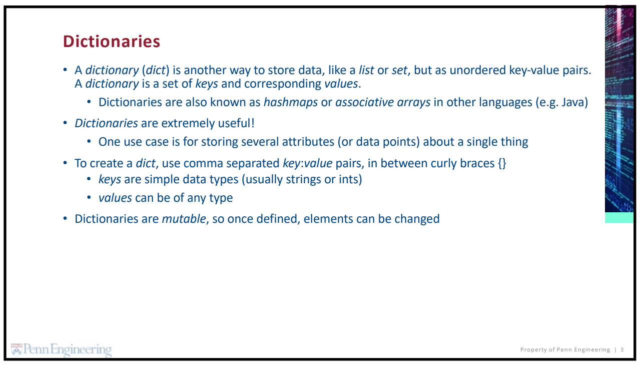
Brandon Krakowsky

1



Dictionaries

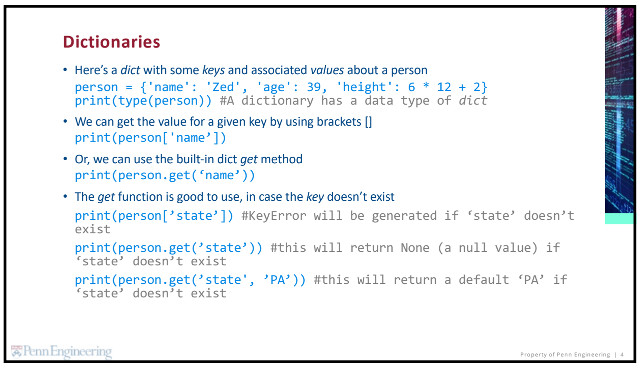
2



Dictionaries

- A *dictionary* (*dict*) is another way to store data, like a *list* or *set*, but as unordered key-value pairs. A *dictionary* is a set of *keys* and corresponding *values*.
  - Dictionaries are also known as *hashmaps* or *associative arrays* in other languages (e.g. Java)
- *Dictionaries* are extremely useful!
  - One use case is for storing several attributes (or data points) about a single thing
- To create a *dict*, use comma separated *key:value* pairs, in between curly braces {}
  - *keys* are simple data types (usually strings or ints)
  - *values* can be of any type
- Dictionaries are *mutable*, so once defined, elements can be changed

3



Dictionaries

- Here’s a *dict* with some *keys* and associated *values* about a person

```
person = {'name': 'Zed', 'age': 39, 'height': 6 * 12 + 2}
print(type(person)) #A dictionary has a data type of dict
```

- We can get the value for a given key by using brackets []

```
print(person['name’])
```

- Or, we can use the built-in dict *get* method

```
print(person.get(‘name’))
```

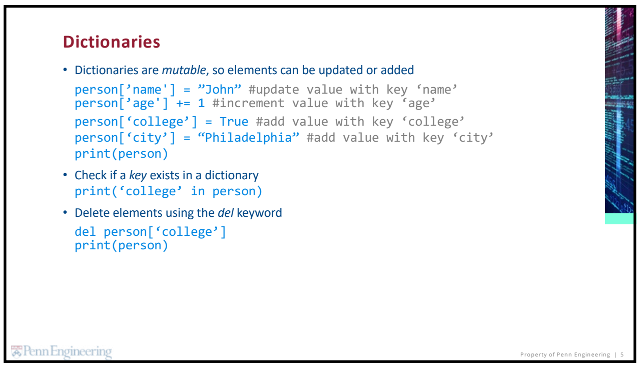
- The *get* function is good to use, in case the *key* doesn’t exist

```
print(person[’state’]) #KeyError will be generated if ‘state’ doesn’t
exist
print(person.get(’state’)) #this will return None (a null value) if
‘state’ doesn’t exist
print(person.get(’state', ’PA’)) #this will return a default ‘PA’ if
‘state’ doesn’t exist
```

Penn Engineering

Property of Penn Engineering | 4

4



Dictionaries

- Dictionaries are *mutable*, so elements can be updated or added

```
person[’name'] = ”John” #update value with key ‘name’
person[’age'] += 1 #increment value with key ‘age’
person[‘college’] = True #add value with key ‘college’
person[‘city’] = “Philadelphia” #add value with key ‘city’
print(person)
```

- Check if a *key* exists in a dictionary

```
print(‘college’ in person)
```

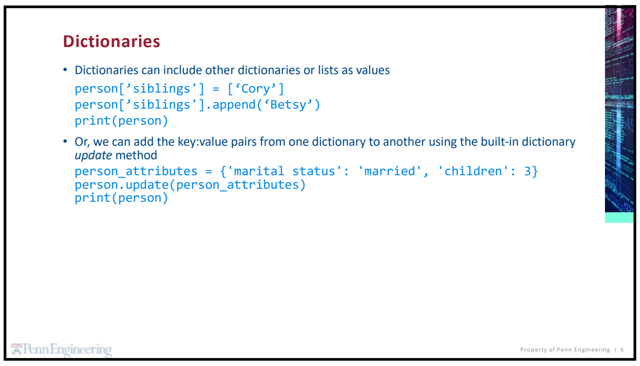
- Delete elements using the *del* keyword

```
del person[‘college’]
print(person)
```

Penn Engineering

Property of Penn Engineering | 5

5



Dictionaries

- Dictionaries can include other dictionaries or lists as values

```
person[’siblings'] = [‘Cory’]
person[’siblings'].append(‘Betsy’)
print(person)
```

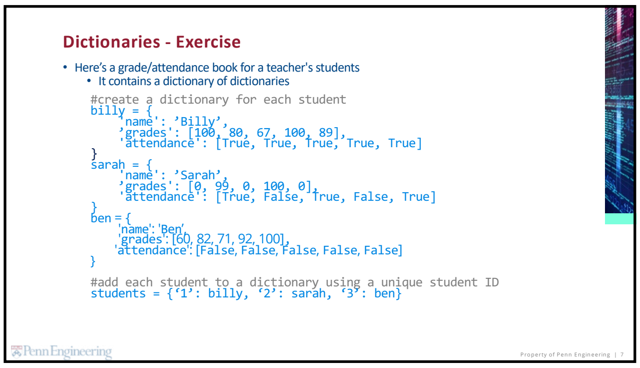
- Or, we can add the key:value pairs from one dictionary to another using the built-in dictionary *update* method

```
person_attributes = {'marital status': 'married', 'children': 3}
person.update(person_attributes)
print(person)
```

Penn Engineering

Property of Penn Engineering | 6

6



Dictionaries - Exercise

- Here's a grade/attendance book for a teacher's students
    - It contains a dictionary of dictionaries

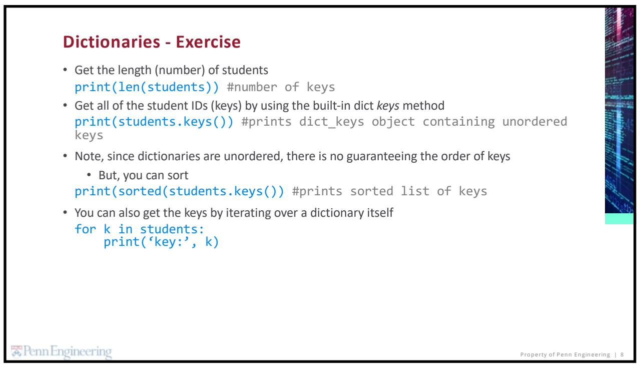
```
#create a dictionary for each student
billy = {
    'name': 'Billy',
    'grades': [100, 80, 67, 100, 89],
    'attendance': [True, True, True, True, True]
}
sarah = {
    'name': 'Sarah',
    'grades': [0, 99, 0, 100, 0],
    'attendance': [True, False, True, False, True]
}
ben = {
    'name': 'Ben',
    'grades': [60, 82, 71, 92, 100],
   'attendance': [False, False, False, False, False]
}

#add each student to a dictionary using a unique student ID
students = {'1': billy, '2': sarah, '3': ben}
```

Penn Engineering

Property of Penn Engineering | 7

7



Dictionaries - Exercise

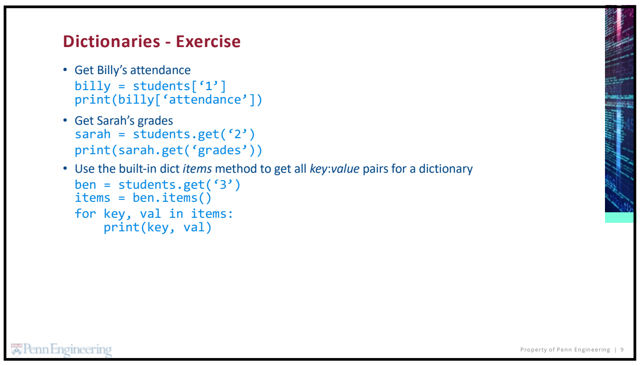
- Get the length (number) of students
  `print(len(students)) #number of keys`
- Get all of the student IDs (keys) by using the built-in dict *keys* method
  `print(students.keys()) #prints dict_keys object containing unordered keys`
- Note, since dictionaries are unordered, there is no guaranteeing the order of keys
    - But, you can sort
  `print(sorted(students.keys())) #prints sorted list of keys`
- You can also get the keys by iterating over a dictionary itself

```
for k in students:
    print('key:', k)
```

Penn Engineering

Property of Penn Engineering | 8

8



Dictionaries - Exercise

- Get Billy's attendance

```
billy = students['1']
print(billy['attendance'])
```

- Get Sarah's grades

```
sarah = students.get('2')
print(sarah.get('grades'))
```

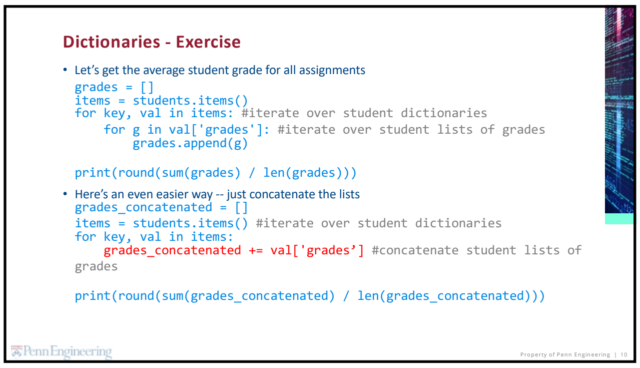
- Use the built-in dict *items* method to get all *key:value* pairs for a dictionary

```
ben = students.get('3')
items = ben.items()
for key, val in items:
    print(key, val)
```

Penn Engineering

Property of Penn Engineering | 9

9



Dictionaries - Exercise

- Let’s get the average student grade for all assignments

```
grades = []
items = students.items()
for key, val in items: #iterate over student dictionaries
    for g in val['grades']: #iterate over student lists of grades
        grades.append(g)

print(round(sum(grades) / len(grades)))
```

- Here’s an even easier way -- just concatenate the lists

```
grades_concatenated = []
items = students.items() #iterate over student dictionaries
for key, val in items:
    grades_concatenated += val['grades’] #concatenate student lists of
grades

print(round(sum(grades_concatenated) / len(grades_concatenated)))
```

Property of Penn Engineering | 10

10