c#

If Statements

University of Wyoming COSC 1010

Adapted from: *Python Crash Course 3rd Ed* By Eric Matthes

Dictionaries

- Now we can talk about Dictionaries, one of Python's most helpful data structures
- These allow pieces of related data to be connected and stored together
- We will learn how to access and modify data within a dictionary
- Dictionaries can store almost limitless amounts of data, so being able to loop through them is also important

Dictionaries

- Dictionaries can be nested within lists
- Lists can be nested in Dictionaries
- Dictionaries can be nested in other dictionaries

Dictionaries

- Understanding Dictionaries enables the modeling of real-world objects more accurately
- For example you could create a dictionary about a person, and store information about them
- With dictionaries any two kinds of information

Working with Dictionaries

- A `dictionary` in Python is a collection of *key-value* pairs
- Each *key* is connected to a *value*
- A key's value can be a number, a string, a list, or even another dictionary
- The value in a dictionary can be any value

Working With Dictionaries

- In python a dictionary is wrapped in braces `{}` with a series of `key:value` pairs inside
  - `states = {'Wyoming':'Cheyenne','Colorado':'Denver'}`
- A `key:value` pair is a set of values associated with each other
- When you provide a key, Python returns its associated value
- Every `key:value` pair is connected with a colon `:`

Working With Dictionaries

- The simplest dictionary contains only one pair
- But, dictionaries can store as many pairs as you want

Accessing Values in a Dictionary

- To get the value associated with a key, give the name of the dictionary and place the key is square brackets
  - `states['Wyoming']`
- That returns the value associated with the key, so `"Cheyenne"`
- You can have an unlimited number of `key:value:` pairs ina dictionary
- You can access any of the values with their corresponding key

In [2]:
```
states = {"Wyoming":"Cheyenne","Colorado":"Denver"}

print(f"The capitol of Wyoming is {states['Wyoming']}")
```

```
The capitol of Wyoming is Cheyenne
```

Adding New Key-Value Pairs

- Dictionaries are dynamic structures, like lists
- You can add new `key-value` pairs to a dictionary at any time
- To add a new `key-value` pair you give the name of the dictionary followed by the key in square brackets, assigning the new value
- So a new state could be added with `states["Montana"] = "Helena"`

In [2]:
```
states = {"Wyoming":"Cheyenne","Colorado":"Denver"}
print(states)
states["Montana"] = "Helena"
```

```
print(states)
```

```
{'Wyoming': 'Cheyenne', 'Colorado': 'Denver'}
{'Wyoming': 'Cheyenne', 'Colorado': 'Denver', 'Montana': 'Helena'}
```

Adding New Key-Value Pairs

- There we defined our states dictionary again
- We printed it out to verify the two states were there
- Then supplying a new key we assign a new value to the list
- Dictionaries retain the order in which they were defined

Starting with an Empty Dictionary

- Sometimes it is helpful to start with an empty dictionary
- Then as needed adding new elements to it
- To start filling an empty dictionary, define a dictionary with an empty set of braces
  - `states = {}`
- Then, each `key-value` pair can be added
- Typically empty dictionaries are used to store suer supplied data

In [3]:
```
countries = {}
print(countries)
countries["United States"] = "Washington D.C"
countries["Canada"] = "Ottawa"
print(countries)
```

```
{}
{'United States': 'Washington D.C', 'Canada': 'Ottawa'}
```

Modifying Values in a Dictionary

- Modifying a value in a dictionary follows the same syntax as adding value
- The dictionary name and key in square brackets with an assignment to it
- You need to be careful to not overwrite when trying to declare

In [4]:
```
colors = {"Red":"#ff0000","Blue":"#0000ff","UW-Gold":"#492f24"}
print(colors)
colors["UW-Gold"] = "#ffc425"
```

```
{'Red': '#ff0000', 'Blue': '#0000ff', 'UW-Gold': '#492f24'}
```

Removing `key:value` Pairs

- When you no longer need information ina dictionary, the `del` statement can be used
- All `del` needs is the name of the dictionary and the key you want to remove
- So to remove a state: `del states['Colorado]`

In [5]:
```
colors = {"Red":"#ff0000","Blue":"#0000ff","UW-Gold":"#492f24"}
print(colors)
del colors["Red"]
print(colors)
```

```
{'Red': '#ff0000', 'Blue': '#0000ff', 'UW-Gold': '#492f24'}
{'Blue': '#0000ff', 'UW-Gold': '#492f24'}
```

Removing `key:value` Pairs

- The `del` statement tells python to delete the key `Red` from the dictionary
- It also removes the value associated with that key
- the deleted `key:value` pair is removed immediately

What to Store in a Dictionary

- Our previous examples were used to store similar kinds of information
  - e.g. States and their Capitals
- You could also use dictionaries to store multiple pieces of information about one object
- Dictionary declarations do not need to declared all out on one line
- You can have new lines between entries

In [11]:
```
university_of_wyoming = {
    "Name": "University of Wyoming",
    "City": "Laramie",
    "State":"Wyoming",
    "Founded":1886,
    "Students":11_100
}
print(university_of_wyoming)
```

```
{'Name': 'University of Wyoming', 'City': 'Laramie', 'State': 'Wyoming', 'Founded':
1886, 'Students': 11100}
```

Using `get()` to Access Values

- Using keys in square brackets to retrieve values has one potential problem:
  - IF the key doesn't exist you will get an error
- Namely a `KeyError`

In [8]:
```
colors = {}
print(colors["Blue"])
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/tmp/ipykernel_38630/3593499717.py in <module>
      1 colors = {}
----> 2 print(colors["Blue"])

KeyError: 'Blue'
```

Using `get()` to Access Values

- Alternatively `get()` can be used
- The `get()` method requires a key as its first argument
- it also takes a second argument optionally, what will be returned if no corresponding key is found

In [9]:
```
colors = {}
print(colors.get("Blue","That key wasn't found"))
```

```
That key wasn't found
```

Looping Through a Dictionary

- Much like with lists dictionaries can hold just a few `key:value` pairs, or millions
- because a dictionary can contain large amounts of data, Python lets you loop through a dictionary
- Dictionaries can be used to store information in a variety of ways
- You can loop through:
  - All of the dictionary's key-value pairs
  - Through its keys
  - Or through its values

Looping Through All Key-Value Pairs

- To begin that's think about the previous UW example
- we already know how to access individual values based on their keys
- But how can we go about looping through all the elements rather than getting them manually
- To do so we can use a `for` loop

In [12]:
```
university_of_wyoming = {
    "Name": "University of Wyoming",
    "City": "Laramie",
    "State":"Wyoming",
    "Founded":1886,
    "Students":11_100
}

for key, value in university_of_wyoming.items():
    print(f"Key: {key}")
    print(f"Value: {value}\n")
```

```
Key: Name
Value: University of Wyoming

Key: City
Value: Laramie

Key: State
Value: Wyoming

Key: Founded
Value: 1886

Key: Students
Value: 11100
```

Looping Through All Key-Value Pairs

- This loop is a *little* different than we have seen
- Here two variables are declared
  - These variables can have any name you want
- The variables function the same as we have seen previously
  - Gaining new values in each iteration through the loop

Looping Through All Key-Value Pairs

- The second part of the `for` loop is also different
- It includes the dictionary name followed by `.items()`
- `.items()` returns a sequence of `key:value` pairs
- The `for` loop then assigns each of these pairs to the two variables

Looping Through All Key-Value Pairs

- Looping though all `key:value` pairs works particularly well for dictionaries storing the

same kind of information

- So the colors example previously
- We know that they `key:value` is always `colorname:hexvalue`
- So, rather than using `key, value` instead `color, hex` could be used

Looping Through All the Keys in a Dictionary

- The `.keys()` method is useful when all the values aren't needed
- So for example, printing the UW dictionary keys would show what attributes are stored
- Doing this would only require a single variable name

In [13]:
```
university_of_wyoming = {
    "Name": "University of Wyoming",
    "City": "Laramie",
    "State":"Wyoming",
    "Founded":1886,
    "Students":11_100
}

print("The attributes available are: ")
for key in university_of_wyoming.keys():
    print(key)
```

```
The attributes available are:
Name
City
State
Founded
Students
```

Looping Through All the Keys in a Dictionary

- The loop tells Python to pull all the keys from the dictionary
- Then, the keys are assigned to the variable `key`
- The output shows all attributes recorded in our list
- Looping through the keys is actually the default behavior when looping through a dictionary
  - `.keys()` just makes it more explicit

In [14]:
```
university_of_wyoming = {
    "Name": "University of Wyoming",
    "City": "Laramie",
    "State":"Wyoming",
    "Founded":1886,
    "Students":11_100
}
```

```
for key in university_of_wyoming:
    print(key)
```

```
Name
City
State
Founded
Students
```

Looping Through All the Keys in a Dictionary

- You can access the value associated with any key within the loop
- This way you can still access both key and value, without explicitly looping through both
- The `.keys()` method isn't just for looping
- It returns a sequence of all keys, and you can then use it to check for keys with an `if`

In [16]:
```
university_of_wyoming = {
    "Name": "University of Wyoming",
    "City": "Laramie",
    "State":"Wyoming",
    "Founded":1886,
    "Students":11_100
}
for key in university_of_wyoming:
    print(university_of_wyoming[key])

if "Elevation" in university_of_wyoming.keys():
    print(f"UW's Elevation is {university_of_wyoming['Elevation']}")
```

```
University of Wyoming
Laramie
Wyoming
1886
11100
```

Looping Through a Dictionary's Keys in a Particular Order

- Looping through a dictionary returns the keys in the order they were inserted
- Sometimes you may want to loop through the dictionary in a different order
- One way to do so is to sort the keys as they are returned in thr for loop
- The `sorted()` function can be used to get a copy of the keys in order

In [17]:
```
university_of_wyoming = {
    "Name": "University of Wyoming",
    "City": "Laramie",
    "State":"Wyoming",
    "Founded":1886,
    "Students":11_100
```

```
}

for key in sorted(university_of_wyoming.keys()):
    print(key)
```

```
City
Founded
Name
State
Students
```

Looping Through a Dictionary's Keys in a Particular Order

- This `for` statement is like the other `for` statement's we've seen
- But, `sorted()` was wrapped around the `dictionary.keys()`

Looping Through all Values in a Dictionary

- Sometimes the values are more important than the keys
- The `.values()` method returns a sequence of the values without any keys
- So suppose you want all the values associated with UW

In [18]:
```
university_of_wyoming = {
    "Name": "University of Wyoming",
    "City": "Laramie",
    "State":"Wyoming",
    "Founded":1886,
    "Students":11_100
}

for value in university_of_wyoming.values():
    print(value)
```

```
University of Wyoming
Laramie
Wyoming
1886
11100
```

Looping Through all Values in a Dictionary

- Here the `for` loop pulls each value from the dictionary and assigns it to the variable `value`
- This approach pulls all the values without checking for repeats
- If you want all values without repeats the `set()` function can be used

- When using `set()` Python identifies all unique items in the collection and builds a set from them

In [19]:
```
programming_languages = {
    "Alice": "Python",
    "Bob": "JavaScript",
    "Charlie": "Java",
    "David": "C++",
    "Eve": "Python",
    "Frank": "Ruby",
    "Grace": "JavaScript",
    "Hank": "Python",
    "Ivy": "Java",
    "Jack": "C#"
}

for language in set(programming_languages.values()):
    print(language)
```

```
JavaScript
C#
Java
Ruby
Python
C++
```

Looping Through all Values in a Dictionary

- As you continue using Python you will discover more and more built in features
- A set can be created manually using braces and declaring elements as you would a list
  - `langauges = {'Python','Julia','Go'}`
- Sets and Dictionaries both use braces, but remember dictionaries have `key:value` pairs

In [20]:
```
lang_list = ['Python','Julia','Go'] #this is a list
lang_tupl = ('Python','Julia','Go') #this is a tuple
lang_set  = {'Python','Julia','Go'} #this is a set
```

Nesting

- Sometimes you'll want to store multiple dictionaries ina list
- Or a list of items as as teh value in dictionary
- This is called *nesting*
  - A dictionary can be nested within a list
  - A list can be nested within a dictionary
  - OR, a dictionary can be nested within another dictionary

A List of Dictionaries

- Previously we had a dictionary for UW, but what if we wanted to store similar information for multiple Universities?
- One way to do so would be to declare each dictionary individually and then place them within a list

In [21]:
```
university_of_wyoming = {
    "Name": "University of Wyoming",
    "City": "Laramie",
    "State":"Wyoming",
    "Founded":1886,
    "Students":11_100
}
colorado_state_university = {
    "Name": "Colorado State University",
    "City": "Fort Collins",
    "State": "Colorado",
    "Founded": 1870,
    "Students": 33_877
}

montana_state_university = {
    "Name": "Montana State University",
    "City": "Bozeman",
    "State": "Montana",
    "Founded": 1893,
    "Students": 16_902
}

universities = [university_of_wyoming, colorado_state_university, montana_state_uni

for university in universities:
    print(university)
```

```
{'Name': 'University of Wyoming', 'City': 'Laramie', 'State': 'Wyoming', 'Founded':
1886, 'Students': 11100}
{'Name': 'Colorado State University', 'City': 'Fort Collins', 'State': 'Colorado', '
Founded': 1870, 'Students': 33877}
{'Name': 'Montana State University', 'City': 'Bozeman', 'State': 'Montana', 'Founded
': 1893, 'Students': 16902}
```

A List of Dictionaries

- There we created three dictionaries, each representing a different university
- Then, each one was stored within the list `universities`
- Finally the list is looped through and the universities are all printed out
- though, this may not be the most realistic way to do it

A List of Dictionaries

- A more realistic example would be code that generates up multiple dictionaries within a list
- Instead of universities, maybe we want to store students
- To begin we could fill a dictionary with default information, to be changed as needed

In [23]:
```
students = [] #decalre our empty list

for student in range(30): #loop through 30 times
    new_student = {'name':"Pistol Pete","email":"pete@uwyo.edu","w#":"W18860307"} #
    students.append(new_student)# add our new student

for student in students[:5]: #go through the first 5 students
    print(student) #print
```

```
{'name': 'Pistol Pete', 'email': 'pete@uwyo.edu', 'w#': 'W18860307'}
{'name': 'Pistol Pete', 'email': 'pete@uwyo.edu', 'w#': 'W18860307'}
{'name': 'Pistol Pete', 'email': 'pete@uwyo.edu', 'w#': 'W18860307'}
{'name': 'Pistol Pete', 'email': 'pete@uwyo.edu', 'w#': 'W18860307'}
{'name': 'Pistol Pete', 'email': 'pete@uwyo.edu', 'w#': 'W18860307'}
```

A List of Dictionaries

- We declared an empty list to hold all the aliens to be created
- The range function returns a series of numbers
  - Telling python how many times to go through
- Each time though a new student is created and appended to the list

A List of Dictionaries

- These students all have the same characteristics, but Python considers each a separate

object
- Then individual students can be modified
- It is common to store a number of dictionaries in a list when each dictionary contains many kinds of information about one object

In [27]:
```
students = [] #decalre our empty list

for student in range(30): #loop through 30 times
    new_student = {'name':"Pistol Pete","email":"pete@uwyo.edu","w#":"W18860307"} #
    students.append(new_student)# add our new student

for student in students[:3]:
    student["name"] = "Cowboy Joe"
    student["email"] = "cj@uwyo.edu"
    student["w#"] = "W03071886"

for student in students[:5]:
    print(student)
```

```
{'name': 'Cowboy Joe', 'email': 'cj@uwyo.edu', 'w#': 'W03071886'}
{'name': 'Cowboy Joe', 'email': 'cj@uwyo.edu', 'w#': 'W03071886'}
{'name': 'Cowboy Joe', 'email': 'cj@uwyo.edu', 'w#': 'W03071886'}
{'name': 'Pistol Pete', 'email': 'pete@uwyo.edu', 'w#': 'W18860307'}
{'name': 'Pistol Pete', 'email': 'pete@uwyo.edu', 'w#': 'W18860307'}
```

A List in a Dictionary

- Rather than putting a dictionary in a list, sometimes you need a list in a dictionary
- Perhaps you want to expand the UW dictionary to include a list of the colleges
- This adds another depth of description to something

In [28]:
```
university_of_wyoming = {
    "Name": "University of Wyoming",
    "City": "Laramie",
    "State":"Wyoming",
    "Founded":1886,
    "Students":11_100,
    "Colleges": ["Agriculture","Arts and Sciences","Business","Education","Engineer
}

print(university_of_wyoming)
```

```
{'Name': 'University of Wyoming', 'City': 'Laramie', 'State': 'Wyoming', 'Founded':
1886, 'Students': 11100, 'Colleges': ['Agriculture', 'Arts and Sciences', 'Business
', 'Education', 'Engineering and Physical Science']}
```

A List in a Dictionary

- We begin with a dictionary that holds information about the colleges at UW

- One key is `Name` which is associated with `University of Wyoming`
- Then the dictionary is printed out
- You can nest a list in a dictionary anytime you want more than one value associated with a key
- You can utilize `print` statements to refine how you interact with the data

In [48]:
```
university_of_wyoming = {
    "Name": "University of Wyoming",
    "City": "Laramie",
    "State":"Wyoming",
    "Founded":1886,
    "Students":11_100,
    "Colleges": ["Agriculture","Arts and Sciences","Business","Education","Engineer
}

for key in university_of_wyoming:
    if type(university_of_wyoming[key]) is list:
        print(f"{key}:")
        for elem in university_of_wyoming[key]:
            print(f"\t\t{elem}")
    else:
        print(f"{key}:    \t{university_of_wyoming[key]}")
```

```
Name:           University of Wyoming
City:           Laramie
State:          Wyoming
Founded:        1886
Students:       11100
Colleges:
                Agriculture
                Arts and Sciences
                Business
                Education
                Engineering and Physical Science
```

A Dictionary in a Dictionary

- Dictionaries can be nested in other dictionaries
- But, admittedly the code can get complicated quickly
- For example if you have users ona website, each with a unique username
- the username could be the key, for a sub dictionary with more information

In [51]:
```
users = {
    "cowboyjoe": {
        'first': "cowboy",
        'last' : 'joe'
    },
    "pistolpete":{
        'first': 'pistol',
        'last': 'pete'
    }
```

```
}

for username, user_info in users.items():
    print(f"username: {username}")
    full_name = f"{user_info['first']} {user_info['last']}"
    print(f"Name: {full_name.title()}\n")
```

```
username: cowboyjoe
Name: Cowboy Joe

username: pistolpete
Name: Pistol Pete
```

A Dictionary in a Dictionary

- Here a dictionary `users` is defined with two `key` values
- the value associated with each key is a dictionary with includes the first and last name
- Then the dictionary `users` is looped though
- Python assigns the key to the variable `username`
- And the dictionary value to `user_info`
- Alternatively you could manually loop though `user_info`

In [52]:
```
users = {
    "cowboyjoe": {
        'first': "cowboy",
        'last' : 'joe'
    },
    "pistolpete":{
        'first': 'pistol',
        'last': 'pete'
    }
}

for username, user_info in users.items():
    print(f"username: {username}")
    for key, value in user_info.items():
        print(f"{key}: {value}")
    print()
```

```
username: cowboyjoe
first: cowboy
last: joe

username: pistolpete
first: pistol
last: pete
```