



Advanced Python

Please take a seat.



Dictionaries

—



Access and Manage Dictionary Data

- Dictionaries are an advanced data structure in Python that consists of multiple key value pairs
- Dictionaries are very useful for relationships between data. Additionally, dictionaries are a very efficient data structure

```
>>> MLB_team = {  
...     'Colorado' : 'Rockies',  
...     'Boston'   : 'Red Sox',  
...     'Minnesota': 'Twins',  
...     'Milwaukee': 'Brewers',  
...     'Seattle'  : 'Mariners'  
... }
```



Getting An Entry

- Retrieving data from a dictionary is very simple
- The two main ways to access a dictionary are shown below
- Note: In a dictionary you always access the value from the key

```
d = {  
    <key>: <value>,  
    <key>: <value>,  
    .  
    .  
    .  
    <key>: <value>  
}
```

```
d = {  
    "apple": "fruit",  
    "celery": "vegetable",  
    "chicken": "meat",  
    "artichoke": "vegetable",  
    "beef": "meat"  
}
```

```
print(d[key])  
print(d.get(key))
```

```
print(d["apple"])  
# fruit  
print(d.get("celery"))  
# vegetable
```



The Difference

- Although both methods of retrieving a value from a key are good, there may be circumstances in which you may want to use one over the other

```
d = {  
    "apple": "fruit",  
    "celery": "vegetable",  
    "chicken": "meat",  
    "artichoke": "vegetable",  
    "beef": "meat"  
}
```

```
print(d["apple"])  
# celery  
print(d["orange"])  
# throws a KeyError
```

```
print(d.get("apple"))  
# fruit  
print(d.get("orange"))  
# none  
print(d.get("orange", "This key does not exist.))  
# This key does not exist.
```

- In most cases, this method will be faster
- Can not handle errors if a key does not exist
- A little bit slower than the other method
- Can handle errors
- If the key does not exist, the method will return None
- Method will return second argument if the key does not exist

Adding New Data into the Dictionary

- Like retrieving data, there are many ways to add data to dictionaries, and each have their own pros and cons

```
print(d)
# {'apple': 'fruit'}

d["orange"] = "fruit"
print(d)
# {'apple': 'fruit', 'orange': 'fruit'}
```

```
print(d)
# {'apple': 'fruit'}

d.update(celery = "vegetable")
print(d)
#{'apple': 'fruit', 'celery': 'vegetable'}
```

Method 1

- This method is simpler and more convenient than the other method
- If this method is used and the key already exists, then the new value will replace the old value
- Used in most cases where you want to add/modify one key value pair

```
d = {  
    "apple": "fruit"  
}  
print(d)  
# {'apple': 'fruit'}  
  
d["celery"] = "fruit"  
# {'apple': 'fruit', 'celery': 'fruit'}  
  
d["celery"] = "vegetable"  
# {'apple': 'fruit', 'celery': 'vegetable'}
```

Method 2

- More versatile than method 1
- Can add more than one key at a time
- Like method 1, overrides key value pairs if it already exists

```
d = {  
    "apple": "fruit"  
}  
print(d)  
# {'apple': 'fruit'}  
  
d.update(celery = "vegetable", steak = "meat")  
# {'apple': 'fruit', 'celery': 'vegetable', 'steak': 'meat'}
```

```
f = {"orange": "fruit", "mushroom": "fungi"}  
d.update(f)  
# d.update({"orange": "fruit", "mushroom": "fungi"})  
# is fine too  
# {'apple': 'fruit', 'celery': 'vegetable', 'steak': 'meat', 'orange': 'fruit',  
  'mushroom': 'fungi'}  
  
l = [("tomato", "fruit"), ("pork", "meat")] # tuples are fine too  
d.update(l)  
# {'apple': 'fruit', 'celery': 'vegetable', 'steak': 'meat', 'orange': 'fruit',  
  'mushroom': 'fungi', 'tomato': 'fruit', 'pork': 'meat'}
```


Not Just Strings!

- A key can be any immutable object - ints, floats, strs, bools, tuples (can not be mutable - list, dictionaries, sets)
- Values can be any object

```
human = {"name": "Jacques",  
         "age": 20,  
         "favorite_foods": ['tomato', 'burger', 'lettuce'],  
         "friends": {"best_friend": "Jerry",  
                      "second_friend": "Finny",  
                      "third_friend": "Jarvis"}  
}
```

Practice

- Explore writing code with dictionaries
- Try adding data, retrieving data, and using the data in a purposeful way
- Let us know if you need any help or need to change slides to a previous slide

Deleting Data from a Dictionary

- There are two main ways to delete data from a dictionary
- Both remove one key at a time, but loops can be used for multiple

```
foods = {  
    "tomato": "fruit",  
    "apple": "fruit",  
    "cherry": "fruit",  
    "chicken": "fruit"  
}  
  
removed = foods.pop('chicken')  
print(removed)  
# fruit  
  
print(foods)  
# {'tomato': 'fruit', 'apple': 'fruit', 'cherry': 'fruit'}
```

```
foods = {  
    "tomato": "fruit",  
    "apple": "fruit",  
    "cherry": "fruit",  
    "chicken": "fruit"  
}  
  
del foods['chicken']  
# {'tomato': 'fruit', 'apple': 'fruit', 'cherry': 'fruit'}
```

Modifying Data?

- You already know how to modify data in a dictionary
- First, retrieve the data using one of the methods
- You might want to store the data in a variable
- Modify the data you obtained
- Then, add the data back using the same key

Shortcut!

- You can modify the data directly
- Treat it like a variable
- Any questions please ask!

```
d['friends'].append("Joe")  
d['age'] += 1  
d['name'] = d['name'][5:]
```

Helpful Hints

`.clear()` - delete all information in a dictionary

`del[yourDict[yourKey]]` - deletes a key value pair from a dictionary

`.pop(yourKey)` - deletes a key value pair from a dictionary - returns the value of the key

`.popitem()` - deletes last key value pair from a dictionary - returns the key value pair in tuple

`.keys()` - returns an iterable with all keys of a dictionary

`.values()` - returns an iterable with all values of a dictionary

`.items()` - returns an iterable with all key value pairs of a dictionary as tuples

Helpful Hints

`.get(key, optional value)` - gets the value of the key in a dictionary, if it does not exist then returns the optional value (defaults to `None`)

`yourDict[key]` - gets value of the key in dictionary, throws error if key does not exist

`d[key] = value` - sets a key value pair in the dictionary

`d.update(anotherDict)` - updates the current dictionary with another dictionary

`key in dictionary` - returns `True` if key exists in dictionary

`(key, value) in dictionary.items()` - returns `True` if key value pair exists in dictionary

`value in dictionary.values()` - returns `True` if value exists in dictionary

Helpful Hints

`max(d), min(d)` - returns max/min key of a dictionary

`max(d, key = d.get), min(d, key = d.get)` - returns key for max/min value of a dict

`dict(sorted(d.items))` - returns dictionary sorted by keys of dict

`dict(sorted(d, key = lambda d, d[1]))` - returns a dictionary sorted by values of dict

`for k in dict:` - loops through all the keys in the dictionary

`for v in dict.values()` - loops through all values of dictionary

`for k, v in dict.items()` - loops through all key value pairs in dictionary

`for i, (k, v) in enumerate(dict.items())` - loops through all key value pairs in dict with associated index



Extracting Text

—



Opening Files

- In Python, you can extract data or text from .txt files and use the data in your programs.
- To extract data from a file, you must read it first.
- To do this, open the file and store it in a variable → `file1 = open('filename.txt', 'r')`
 - This "r" means you have "reading access" to the file. You can read what's in the file, but you can't write to it
- **IMPORTANT:** the text file must be in the same folder as the program!
- `file2=open('folderName/filename.txt', 'r')` for inside folders
- **NOTE:** At the end of each line in a text file, there is a `'\n'`
- **NOTE:** `'\t'` means that there is a tab between information



Reading Files

- From here, there are different ways to actually read the .txt file and store what is read in different variables.
- `.read()` - reads all of the text from a file and returns it as a string
- `.readline()` - reads the first line of the text file
- `.readlines()` - reads the text line by line and returns each line into a string, which are all contained within a list
- `.split()` - splits the text file into a list separated by a certain parameter

Examples

textFile.txt ×

```
1 Gerry   Atrick  56  75  49  44  79
2 Dee Light  90  89  91  99  97
3 Maureen Corp  76  76  78  74  90
```

Files

Other Stuff

function.py

other_main.py

main.py

textFile.txt

Packager files

main.py ×

```
1 newFile=open('textFile.txt','r')
2
3 readList=newFile.read()
4
5 print(readList)
```

Console Shell

```
Gerry   Atrick  56  75  49  44  79
Dee Light  90  89  91  99  97
Maureen Corp  76  76  78  74  90
```

main.py ×

```
1 newFile=open('textFile.txt','r')
2
3 readList=newFile.readline()
4
5 print(readList)
```

Console Shell

```
Gerry   Atrick  56  75  49  44  79
```

```
>
```

Examples (2)

```
main.py ×
1 newFile=open('textFile.txt','r')
2
3 readList=newFile.read().split('\n')
4 |
5 print(readList)
```

Console Shell

```
['Gerry\tAtrick\t56\t75\t49\t44\t79', 'Dee\tLight\t90\t89\t91\t99\t97', 'Maureen\tCorp\t76\t76\t78\t74\t90']
> |
```

```
main.py ×
1 newFile=open('textFile.txt','r')
2
3 readList=newFile.readlines()
4 |
5 print(readList)
```

Console Shell

```
['Gerry  Atrick  56  75  49  44  79\n', 'Dee  Light  90  89  91  99  97\n', 'Maureen  Corp  76  76  78  74  90']
> |
```



Tables

—



String Formatting/Zones

- Zones are used to create tables
 - There are two different ways to use zones.
 - With a variable
 - Typed as follows: `variableName="{0:^10s}{1:>10d}"`
 - The curly brackets specify the start and end of the zone
 - The colon separates the specific data about the zone from the zone number
 - `^`, `>`, `<`, arrows are used to specify whether you want the data aligned to the center, right, or left. If no arrow is placed in, then string will automatically format to the left, integers will automatically format to the right, and floats will automatically format to the right.
 - Then the next number specifies exactly how many spaces that you want the zone to have. If no number is typed afterwards, then it will only use the amount of spaces that it needs.
 - The s, d, or f after the number specifies what type of data it is. The s is for string. The d is for integer. The f is for a float.
 - Then when you print the data using a print statement, add a `.format()` at the end.
 - Without a variable
 - You only need the zone number, and a `.format()` at the end. That's it!
 - `print("{0} love Hack the Ram!".format('I'))`



Example of String Formatting/Zones

main.py ×

```
1  strFormat="{0:^10s}{1:^10d}"
2
3  print(strFormat.format('name',10))
4
5  print('-'*75)
6  print('Hi my name is {0} and I am {1} years old.'.format('Joe',15))
```

Console Shell

```
name      10
```

```
-----
Hi my name is Joe and I am 15 years old.
```

```
>
```




Examples of How to Make a Table

main.py ×

```
1 strFormat="{0:^10s}|{1:^10d}|"      #this creates a zone, that is centered, that is used to format the table
2 strFormat2 = "\\033[4m{0:^10s}\\033[0m|\\033[4m{1:^10s}\\033[0m|"      #this just underlines the table headers
3 nameList=("Jacob", 'Jeff', 'Joey', 'Jack', 'Jean') #this is a list of all of the names of the people
4 ageList=(10, 10, 13, 18, 22)      #this is a list of all the ages of the names
5
6 print(strFormat2.format('Name','Age')) #this prints out the title of the table
7
8 #this is the loops that will display the entire data table information onto the screen
9 for i in range(len(nameList)):
10     print(strFormat.format(nameList[i], ageList[i]))
11
```

Console Shell

Name	Age
Jacob	10
Jeff	10
Joey	13
Jack	18
Jean	22





Classes

—



Initializing a Class

- A class is the blueprint of an object in Python. You have probably used many of built-in classes without knowing it.
- Classes can help organize data and perform operations. They act as objects for a much bigger project.
- A class will almost always contain an `__init__()` method which initializes the values needed for the class.
- A class can contain methods which are functions that affect that class, and attributes which are values that the class contains.
- Python is an object-oriented programming language.



Example: Lists

- Lists are data structures that contain multiple elements. However, they are actually a built-in class for python.
- A list has methods which alter the data within it. For example, the `.sort()`, `.reverse()` and `.pop()` methods are all part of a list.
- The methods all change the values within the list, and can't affect anything outside of it.



Methods and Attributes

- Methods and attributes are functions and variables that are specific to a class
- Each instance will act independently of any other (ex. If you have 2 of the same type of instance, they might not have the same values)
- Methods and attributes of an instance can only be accessed through the instance



Guided Practice: Cars



```
> 99 little bugs  
in the code,  
> 99 little bugs.  
>  
> take one down,  
> patch it around.  
>  
> 200 little bugs  
in the code..
```

Others : why do you always
use i,j variabes in loops?

Programmers :



Walk through creating a class about cars on adjacent board



Self

- Each class has a self parameter which is used for all of the attributes of a class
- Putting a self before a variable name means that it is specific to that class
- Self represents all of the methods and attributes of an instance of a class
- Self is not a python keyword, but it is general convention to use this word (you can use any other variable name to represent self).



Example: Lemonade Shop

[Link](#)



Steps to creating a class

1. Define a class by using the keyword `class`, the name of your class, and a colon.

```
class Car:
```

2. Create an `__init__()` function inside your class. Pass in what variables and values are necessary.

```
class Car:
    def __init__(self, model, year, brand, gas):
        self.model = model
        self.year = year
        self.brand = brand
        self.gas = gas
```

3. Use your class and make whatever methods and attributes you want

```
class Car:
    def __init__(self, model, year, brand, gas):
        self.model = model
        self.year = year
        self.brand = brand
        self.gas = gas
        self.tank = 20
    def information(self):
        print(f"This {self.brand} {self.model} car was made in {self.year}")
    def drive(self, amount_droven):
        self.gas -= amount_droven
    def refill(self, amount):
        self.gas += amount
```



Hints

```
my_grocery_store = GroceryStore()  
my_grocery_store.sell_items()  
my_grocery_store.restock()
```

```
def sell(self, item):  
    cost = self.check_price(item)  
    self.money += cost
```

- Methods can be called outside of a class by using the class name and method name.
- Methods can be called inside of a class by using the keyword self and the method name.
- When creating a method of a class, you need to pass in the variable self (must be first variable passed in). When calling a method, you do not need to pass it in.
- Don't overuse the self variable. Only use it for attributes that you want to keep.
- A class can be called multiple times, and each class is independent from the other.



Hints

- Attributes can be called outside of a class by using the class name and method name.
- Attributes can be called inside of a class by using the keyword self and the method name.
- All of your attributes are accessible through the self variable.
- Classes should only handle themselves, and what is passed into them.
- Your main code should be able to handle your classes.
- Methods are like functions, attributes are like variables.

```
def sell(self, item):  
    cost = self.check_price(item)  
    self.money += cost
```



Questions?



Practice

- Create a student class that has attributes of first name, last name, age, grade, id (random 5 digit number), classes (list), and grades from a text file
- Add a method that calculates the average of all their grades and returns it
- Add a method that can add a class, remove a class, add a grade, and remove a grade
- Add a method to display all information about the student (name, age, grade, id, classes, and average grade) using zones
- Make a table comparing last name vs grade
- Add a method to determine who had the best grade
- Add a method that will determine who has F's in any class, that returns their name and the class that they have an F in

Data

Name	Age	Grade	ID	Classes	Grades
------	-----	-------	----	---------	--------

Joe Marino	18	12	65432	[Trigonometry, Calculus BC, German 5]	[92, 100, 56]
------------	----	----	-------	---------------------------------------	---------------

Jack Youginer	17	11	45876	[Calculus AB, Physics, Spanish 5]	[88, 90, 89]
---------------	----	----	-------	-----------------------------------	--------------

Mark Rougner	17	12	32785	[Chemistry, Computer Science, German 4]	[99, 62, 89]
--------------	----	----	-------	---	--------------

Cam Waggy	16	10	57954	[Art, European History, Computer Science]	[77, 82, 100]
-----------	----	----	-------	---	---------------

Inkey Quiltner	15	9	21846	[Programming 101, Spanish 1, French 1]	[73, 54, 82]
----------------	----	---	-------	--	--------------