COMP 1510 Winter 2022

Programming Methods

Week 13: File IO, intro to classes, RDD

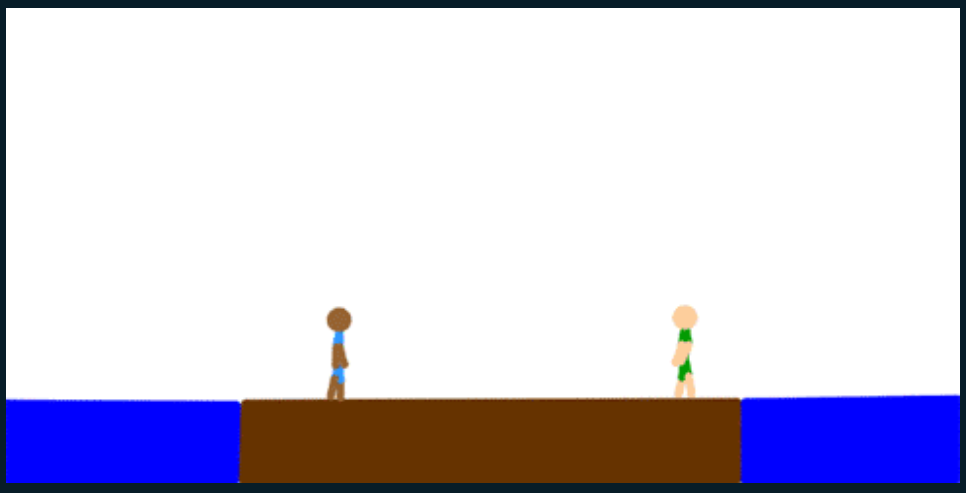
1

Agenda for week 13

1. File IO: opening, reading from, writing to, closing, deleting files
2. Context managers and else blocks
3. Context managers and file-like objects
4. Working with JSON
5. Intro to classes
6. State, attributes and class-level variables
7. Instance initializers, validation and invariants
8. Methods
9. Classes vs objects
10. Designing good classes
11. Unit testing classes

2

From my “gifs you can hear” collection…



3

FILE IO

4

What kinds of files are there?

- Lots
- Text files, binary files, music files, videos, word processing docs, presentations, etc.
- Text files only contain Unicode characters
- All other files formats contain formatting information specific to that file format
- We usually need a special kind of program to open special kinds of files, i.e., we need Word to open docx files
- *Underlying everything is the byte. In files, everything is a byte. Don't forget this. A file is just bits and bytes. 0s and 1s. All files.*

5

Text files (files containing Unicode characters)

- Contain no style information
- Contain only human-readable characters
- (for the most part, anyway, though we know there are some hidden ASCII and Unicode characters)
- Occupy very little space
- We can open a text file in any text editor and read it
- Our Python source code, for example, is a plain text file with a special filename extension (.py)

6

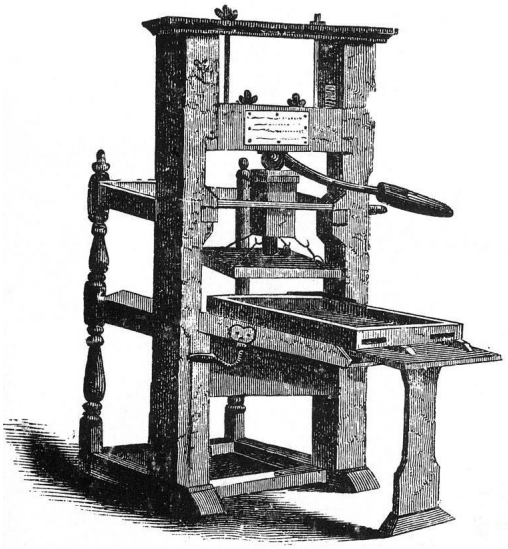
Project Gutenberg

The gold standard of text files is the repo at Project Gutenberg

It's a pre-internet open-source project

Plain-text file versions of great written classics

https://www.gutenberg.org



7

Let's get to it: I want to open a file

1. Navigate to the folder containing my target file
2. Start python3
3. Execute the following:

```
>>> with open('pi_digits.txt') as file_object:
...     contents = file_object.read()
...     print(contents)
...
3.1415926535
  8979323846
  2643383279
```


8

The open( ) function

- Accepts a str argument: the name of the file to open
- Python will search for this file in the directory that contains the Python script being executed
- **Returns an object representing the file**
- Note we did not call close( ) to close the file
- We used the *with* keyword instead

The keyword ***with*** uses *context managers*

- A *context manager* is any Python object that defines a *runtime context* to be established when executing a with statement
- The with statement encapsulates the execution of a block of code and quietly uses two special methods defined by a context manager
- These methods are used to *manage a resource*
- Managing a resource means *opening it, controlling access to it, and closing it*

Using with to open and close files

- Using with, we can use anything that returns a context manager (like the built-in open() function)
- Very conveniently closes the file once access to it is no longer needed
- We don't need to worry about closing the file
- It can be challenging to know when to close the file
- If we forget to close it, that's a problem (it can cause data to be corrupted)
- The "file descriptor" is closed when we leave the with block.

11

The read( ) method

- Belongs to the TextIOBase class of objects which are "text streams"
- We will talk about classes in a few minutes
- For now, we can consider the file object returned by open( ) to be a special kind of TextIOBase object
- We can read the content of a TextIOBase object using read( )
- The read( ) function returns a string containing the entire file contents

12

What if we want to read line by line?

- Sometimes we don't want to dump an entire file into memory at once (especially if it's enormous)
- We may wish to open and search through the file or examine it without saving it to memory
- We can use a for loop on the file object to examine it line by line by line:

```
filename = "pi_digits.txt"
with open(filename) as file_object:
    for each_line in file_object:
        process(each_line) # or whatever else we want to do
```


13

Making a list of lines from the file

- When we use *with* the file object returned by open( ) goes out of scope and is automatically closed when we exit the with-block
- To retain access to the file contents outside the with-block, we need to read and store the file's lines in a list outside the block and then work with that list

```
filename = "pi_digits.txt"
with open(filename) as file_object:
    lines = file_object.readlines()
for line in lines:
    # process each line from the file etc.
```


14

What about large files?

- No different!
- pi_million_digits.txt contains (wait for it) the first million digits of pi
- We can create a single string containing all of it:

```
filename = 'pi_million_digits.txt'
with open(filename) as file_object:
    lines = file_object.readlines()
pi_string = ''
for line in lines:
    pi_string += line.rstrip() # But why do this?!
```


15

So far, we've done the I in IO. What about O?

- How do we write to a file?
- We invoke open and *pass a second argument* that indicates we wish to write to a file:

```
filename = 'programming.txt'
with open(filename, 'w') as file_object:
    file_object.write('I love Python.')
```


16

What if we want to write multiple lines?

- The write method does not add newline characters
- We must append a newline \n after each write to a file
- A new file will be created if it doesn't exist yet:

```
filename = 'programming.txt'
with open(filename, 'a') as file_object:
    file_object.write("JavaScript is okay.\n")
    file_object.write("Python is better, tho.\n")
```


17

File modes

We can open a file in one of several modes:

1. "**r**" is read-mode (default)
2. "**w**" is write-mode (deletes and overwrites)
3. "**a**" is append-mode (doesn't delete)
4. "**r+**" is read AND write mode

Omitting the argument opens in read-only mode "r" by default (safe!)

18

Text file methods summary

1. *read(size=-1)* reads *size* characters or, if no size is specified, to the end of the file (EOF)
2. *readline(size=-1)* reads *size* characters or until newline or EOF
3. *readlines( )* reads and returns a list of lines from the file
4. *write(s)* writes the string s to the stream and returns the number of characters written
5. *seek(offset)* next 2 slides
6. *tell()* next 2 slides

19

The file cursor

- Python maintains a file cursor
- It 'points' to a location in the file
- We use the cursor to track our position in a file (stream)
- The cursor starts:
  - at the beginning of the file when opened in read or write mode
  - At the end of the file when opened in append mode
- We can track and modify the file cursor's location in a file using the tell and seek methods that belong to streams.

20

tell( )

- **Returns** the current stream location as an integer
- We can store this location and return to this spot in the file using the seek function

seek(offset)

- **Moves the cursor** to the specified offset
- Returns the new current stream location as a number
- seek(SEEK_SET) moves the pointer to the beginning of the stream
- seek(SEEK_END) moves the pointer to the end of the stream

21

Favour the idiom

Recall:

1. read( ) reads and copies the **entire** file into a **string**
2. readlines( ) reads the **entire** file and makes a **list of strings** from it
3. readline( ) reads and copies a **single line** into a **string**

When the file cursor is at the end of the file, the read, readlines, and readline methods will all return an empty string

```
with open(filename) as file_object:
    for line in file_object:
        # and so on
```


22

Do you remember StringIO

- Python provides a class called StringIO in the io module
- It can be used as a mock open file
- We can read from it using regular file-reading techniques as if it were a regular file
- We can use it anywhere we'd need a TextIO object
- Examine total.py and total_stringio.py
- This should remind you of our unit tests!

Finally: how do we delete a file?

- It's easy as pi(e):

```
import os

# Delete favourite_soups.txt
os.remove('/Users/c/special_files/favourite_soups.txt')
```

Asserting a file exists

```
import unittest
import pathlib

class TestCase(unittest.TestCase):
    def test(self):
        # ...
        path = pathlib.Path("a/b/c.txt")
        self.assertTrue(path.is_file())
        self.assertTrue(path.parent.is_dir())

if __name__ == "__main__":
    unittest.main()
```


25

QUIZ TIME!

1. There are four plaintext files on Slack:
   1. alice.txt
   2. little_women.txt
   3. moby_dick.txt
   4. siddhartha.txt
2. Choose one of these files
3. How many times does the letter **q** appear in it?
4. How many times does the word **the** appear in it?

26

QUIZ TIME!

Create a file called file_io.py. Inside write a program that:

1. **asks** the user to enter a plaintext file file name
2. **opens** the file if it exists, and creates the file if it doesn't
3. **prints** the contents of the file (if there is anything)
4. **asks** the user for a line of input
5. **appends** the line of input to the end of file (so running the program multiple times will result in a file with multiple lines of input in it)
6. **closes** the file and the program.

27

JSON AND DATA PERSISTANCE

28

JSON (JAY-sahn -sun)

- JavaScript Object Notation
- Originally developed for JavaScript (go figure!)
- Now a common format used by many languages and frameworks

```
{
  "firstName": "Chris",
  "lastName": "Thompson",
  "isAlive": True,
  "age": 38,
  "address": {
    "streetAddress": "West Georgia Street",
    "city": "Vancouver",
    "province": "BC",
    "postalCode": "A1A B2B"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "604-555-1234"
    },
    {
      "type": "office",
      "number": "604-555-5678"
    },
    {
      "type": "cell",
      "number": "604-555-1234"
    }
  ],
  "children": [ ],
  "spouse": … and so on, and so on…
}
```


29

Data persistence with JSON

- Storing data for later
- At BCIT you will learn how to use databases like MySQL and Firebase
- We can use the json module in Python to "dump" a Python dictionary into a JSON data object in a file
- We can "load" the JSON data object from the file later
- We can use JSON to share data between programs and program instances
- Since it's not specific to Python, we can share data with other systems that use other languages

30

We're already JSON experts

- *It's just a dictionary!*
- JSON can't store every kind of Python value
- JSON cannot represent Python objects
- JSON can only store these data types from Python:
  1. Strings
  2. Integers
  3. Floats
  4. Booleans
  5. Lists
  6. Dictionaries
  7. None (stored as *null* in JSON)

Sending (dumping) JSON data to a file

1. Import json
2. Use the json.dump function which takes 2 parameters:
   i. the data to store
   ii. a file object to store the data:

```
import json
numbers = [2, 3, 5, 7, 11, 13]
filename = 'numbers.json'
with open(filename, 'w') as file_object:
    json.dump(numbers, file_object)
```

Loading JSON data from a file

- Even easier!
- Use the json.load function:

Examples:
1. remember_me.py
2. input.py

```
import json
filename = 'numbers.json'
with open(filename) as file_object:
    numbers = json.load(file_object)
print(numbers)
```


33

Read a json-formatted string with loads( )

```
string_of_json_data = '{"name": "Larry", "isCat": true,
                        "miceCaught": 0, "felineIQ": null}'

import json
json_data_as_python = json.loads(string_of_json_data)
print(json_data_as_python)

{'name': 'Larry', 'isCat': True, 'miceCaught': 0, 'felineIQ': None}
```


34

Write a json-formatted string with dumps( )

```
python_dictionary = {'isCat': True, 'miceCaught': 0,
                              'name': 'Zoë', 'felineIQ': None}

import json
string_of_json_data = json.dumps(python_dictionary)
print(string_of_json_data)

{"isCat": true, "miceCaught": 0, "name": "Zoë", "felineIQ": null}
```


Activity Time!

- Let's get the current weather from an online resource
- We can use OpenWeatherMap.org which has a free API
- We can decompose this task into steps:
  1. Download JSON weather data from OpenWeatherMap.org
  2. Ensure the JSON data is in a Python data structure we can use
  3. Search for the forecast in the data (this can be tedious)
  4. Print the forecast.
- How can we do this in Python?

OpenWeatherMap.org

- Small IT company established in 2014
- A group of engineers and experts in Big Data, data processing, and satellite imagery processing
- HQ is in the UK, the development team is in Latvia
- A subscription and API key are free
- Check it out: https://openweathermap.org/api
- Sign up here: https://home.openweathermap.org/users/sign_up

Python approach

1. Use requests.get( ) to download the JSON from the website
2. Use json.load( ) or json.loads( ) to convert the JSON data to something we can use
3. Extract the data we want (tedious, but required)
4. Print the result.

Tangent: the requests module

- Easy way to download files from the internet
- Abstracts away issues like network errors, connection problems, data compression, etc.
- Doesn't come with Python
- Must use pip to install
- Pip is the Python package manager
- Or PyCharm will do it for you too!
- Check out playing_with_requests.py

Step 1: download the json data

```
import requests

url = "http://api.openweathermap.org/data/2.5/forecast?id=6173331&APPID= 73e6ddxxxxxxxxxx0ac198ce"
response = requests.get(url)
```

- Vancouver is city number 6173331 at OpenWeatherMap.org
- I got an API key: 73e6ddxxxxxxxxxx0ac198ce
- API call is api.openweathermap.org/data/2.5/forecast?id=6173331&APPID=MYKEY

Step 2: convert using json module

```
import json

vancouver_weather = json.loads(response.text)
w = vancouver_weather['list']
```

- The response text is a JSON object in string format
- We can convert the string-based JSON object to a Python dictionary

(Yes, that's all there is to it! Isn't abstraction wonderful?)

Step 3: Extract and print the weather

- We must parse the file to identify what we want to print
- Warning: OpenWeatherMap.org gives us a LOT of information!
- Hint: Check out the value associated with the key 'list':

```
wow = vancouver_weather['list']
print('Current weather in Vancouver:')
print(wow[0]['weather'][0]['main'], '-',  \
      wow[0]['weather'][0]['description'])
```

INTRO TO CLASSES

OMG FINALLY

43

Classes and OOP

- So far, we have been using procedural programming
- The main unit of code in procedural programming is the function
- Procedural programming is great, but…
- *OOP (object-oriented programming) is fun, too*!
- In OOP we define new data types that represent real-life things:
  - Define attributes to store the state
  - Define behaviours (write methods) that act on the state
  - "Instantiate the class" to make an object and use it (just like Python!).

44

A class is just another data structure!

- It's a very powerful concept you will use in many Python libraries
- (That's why we are talking about this now)
- Great modularization
  - **Easy** to understand
  - **Easy** to collaborate
- Great for modeling real world problems
- **Encapsulates** data and the functions that work on it
- One of the **dominant** programming paradigms today.

How?

Check out die.py!

- Define classes to represent data types with well-defined characteristics and functionality
- A class serves as a template for creating independent, distinct objects in memory
- An object, as you know, has state and behavior and type
- For a Die:
  - State is the face that is showing
  - Its primary behaviour is that it can be rolled
  - We represent a die by designing a class called Die that models this state and behavior
- We can make as many instances, i.e., as many dice, as we want!

How?

- A class define what sorts of variables an *instance* of a class contains
- These values define the state of an object (instance) of a class
- The methods (functions defined inside the class) define the behaviours
- We design our classes to be versatile and reusable:
- For example:
  1. don't hard-code the number of sides on a die
  2. let the user construct a die, i.e., instantiate an object, with the number of sides desired.

How about a dog?

Check out dog.py!

__init__(self, name, age) method

a) Special function
b) Executed every time we create a new instance of a class
c) Note the leading and trailing dunders
d) Accepts three parameters:
   a) self – every method call associated with a class must pass self, which is a reference to the instance against which the function (method) is being invoked
   b) name
   c) age
e) Note that the variables defined in the __init__ method are prefaced with self
f) Any variable prefaced with self is available to every method in the class, i.e., it has instance-level scope!

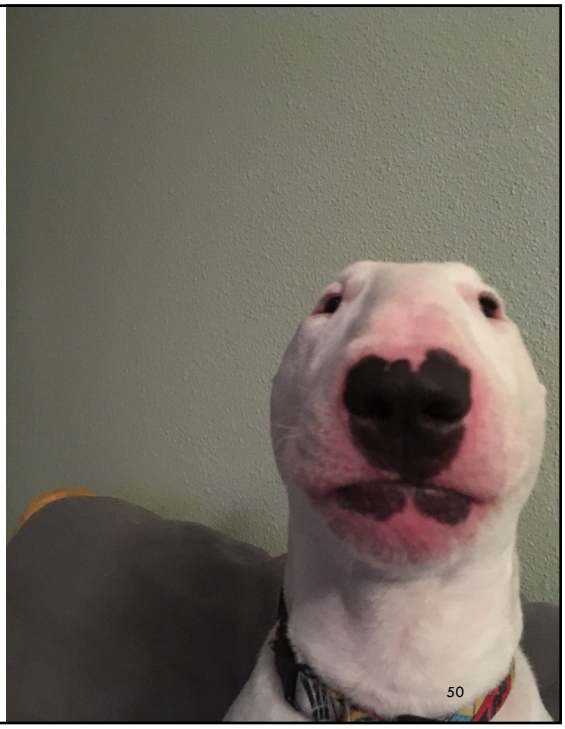
State

- The class source file defines what instance variables (data) an object contains
- Each object has its own unique attributes (values) stored in its instance variables
- The state of each object is the values in these variables
- These variables are an object's private property and last its lifetime
- State can change – we can mutate the state of the object
- *Mutating the state of an object means we are changing the values in its instance variables*

49

More dog behaviours

1. The sit( ) method simulates a dog sitting.
2. The roll_over( ) method simulates a dog rolling over
3. Neither method requires arguments, so the only parameter is self
4. Every dog we create (instantiate) can independently sit and roll over.



50

Methods are just functions

- Classes define and implement **methods** that objects can invoke
- We **invoke or call** methods to communicate with objects or cause them to behave in a specific way
- Methods may require **parameters** that are used to pass **arguments** (input) needed for the behaviour
- Some methods calculate or produce a result called the **return value**
- A class usually provides an **interface of methods** that permit an object to interact with its environment and change state

Making an object from a class

- We call this *instantiation*
- We are creating an instance of a class
- That instance has its own location and address in memory
- It is an independent dog object unrelated to other dog objects
- We can make multiple instances of a class, i.e., we can make dogs
- In this example, my_dog and your_dog are two dog objects that have no knowledge whatsoever of one another:

```
my_dog = Dog('Jayden Dior Fierce', 6)
your_dog = Dog('Heidi N Closet', 3)
```

Accessing attributes, invoking methods

To access the value of an attribute, we can use *dot notation*:

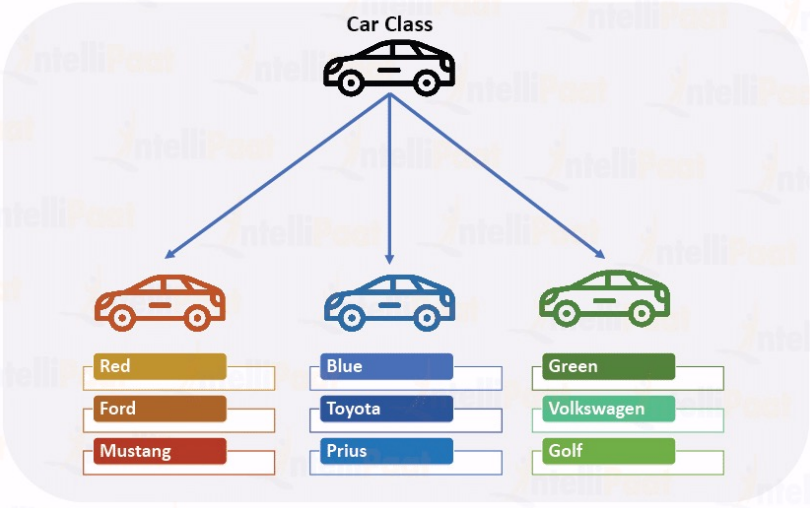
```
my_dog = Dog('willie', 6)
name = my_dog.name
```

To use/invoke/call a method, we use *dot notation*:

```
my_dog = Dog('willie', 6)
my_dog.sit()
my_dog.roll_over()
```


Classes vs objects: some examples

1. first_class.py
2. using_a_class.py
3. die.py
4. dog.py
5. car.py
6. circle.py
7. point.py
8. enumfun.py



CLASS LEVEL VARIABLES

55

Class variables

- Class variables are variables that are *shared by all instances of a class*
- Some languages call these static variables
- Recall that instance variables are private and unique to an object
- Each dog has its own name, its own age, its own dental records
- All objects of a type share a class variable (not unique)
- It is declared inside the class but not inside a method
- It can be accessed using ClassName.class_variable
- Check out cat.py and staticdemo.py

56

DESIGNING GOOD CLASSES

57

The concept of a class

- A class is a blueprint that defines a type, e.g., "Dog," "TextIO," "Integer"
- A class should go into its own module
- The code in a class file describes the data type:
  - What sort of data it contains (its state)
  - How to construct it (the __init__ method)
  - How to interact with it (its behaviours)

58

We use classes to make objects

- We call a Python program that uses classes an '**object-oriented**' (OO) program
- It contains one more more objects working together
- Creating an object is called instantiation, i.e., we are creating an instance of a class
- Many object **instances** (unique versions) can be created from a single class
- Each instance is independent and occupies its own memory space

Designing classes (and functions, too!)

- Design before you code!
- Every class needs to be well-defined:
  1. Represents a single clear concept
  2. Maintain information by storing data in instance variables
  3. Perform actions by executing code in their methods and modifying the state of the program
  4. Don't duplicate data
  5. Don't store more than we need
  6. Minimize "moving parts"
- Employ Abbot's heuristic to identify classes and attributes

Abbot's heuristic

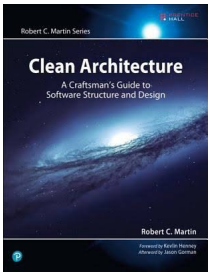
Table 5-1 Abbott's heuristics for mapping parts of speech to model components [Abbott, 1983].

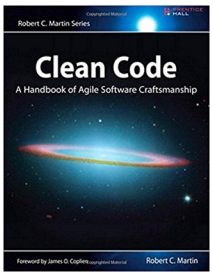
Part of speech	Model component	Examples
Proper noun	Instance	Alice
Common noun	Class	Field officer
Doing verb	Operation	Creates, submits, selects
Being verb	Inheritance	Is a kind of, is one of either
Having verb	Aggregation	Has, consists of, includes
Modal verb	Constraints	Must be
Adjective	Attribute	Incident description

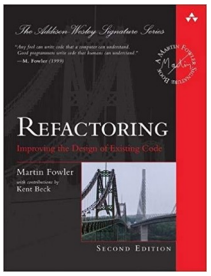
This is from a very interesting document at https://dademuch.com/2017/11/09/uml-analysis-basics/

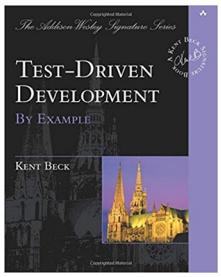
Aside: important books

- There are some "very important books" about programming and software development
- Some contain very important guidelines we try to follow
- You will encounter most and you should read some:





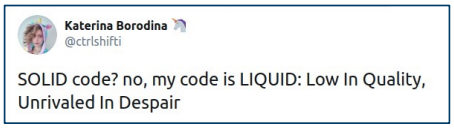




SOLID principles (Robert C Martin)

1. **S**RP **Single responsibility principle**
2. **O**CP Open-closed principle
3. **L**SP Liskov substitution principle
4. **I**SP Interface segregation principle
5. **D**IP Dependency inversion principle

There are many more principles, guidelines, and heuristics, i.e., the Law of Demeter, GRASP, etc.



63

Single responsibility principle

- **A function should do one, and only one, thing**
- **A class should manage one thing**
- Gather the things that change for the same reason
- Separate things that change for different reasons
- Consider a module that compiles and produces a report
  - It can be changed in two ways:
    1. The report content can change (substantive)
    2. The report format can change (cosmetic)
  - These two aspects are different problems with different solutions and will end up in two different modules.

64

Public vs private

- We like to *expose a public interface*
- The interface is how we interact with an object
- *But some things should be private!*
- We say the *implementation is private*
- An object's data is conceptually private
- How can we enforce privacy in Python?

65

The dunder prefix means private in a class

- We enforce privacy with *conventions*
- In Python, we preface private function and instance variable identifiers with dunder, i.e., __count
- A private method is only used by other methods in a class
- It is not intended to be visible or accessible outside the class
- A private variable is only used inside the class, or the class methods
- It is not intended to be visible or modifiable from outside the class

66

Encapsulation is not fool-proof

- Encapsulation prevents accidental but not intentional access
- Private attributes are not really hidden, it's just a message to other developers
- We can still access them if we want. We're not supposed to. But we can…

```
class Cat:
    def __init__(self, age):
        self.__age = age
…
my_cat = Cat(5)
my_cat.age = 5 # ERROR NO NO NO
my_cat._Cat__age = 5 # Works, but I would not hire you
```


OOP is just abstraction

How do we use abstraction?

1. Divide a problem into sub-problems
2. Divide a sub-problem into sub-sub-problems
3. Keep doing this until the individual problems are small enough to solve
4. **Solve the small problems**
5. Ignore the details of the solutions and treat each one as a single encapsulated building block (abstract the details away)

OOP is all about modularization

Abstraction uses modularization.

Modularization is the process of dividing something into well-defined parts

Each of these modules:

1. Is built separately
2. Encapsulates data with the methods that act on it
3. Can be examined separately
4. Interacts in well-defined ways.

69

Abstraction + Modularization

We write code that is modular:

- Functions
- Modules
- Classes
- Methods

We use abstraction by working with components without worrying about their details:

- Implementation separate from public interface
- Encapsulation
- We call this *responsibility-driven design*

70

UNIT TESTING CLASSES

71

We are unit testing experts (well, soon)

- We know every unit test is composed of **three steps**:
  1. Assemble
  2. Act
  3. Assert
- We know every unit test has a very **specific name**, and tests a unit in a very specific way
- Unit tests **must pass**
- Unit tests are *created to ensure our functions work with all disjointed equivalency partitions.*

72

How do we test classes

- Very easily!
  1. **Create** a unit test file for the class
  2. **Test** ALL the methods for the class in this one single file.
- The unit test file will contain multiple unit tests for multiple methods
- Note:
  - Each unit test will need to instantiate the class being tested
  - There's a **shortcut** for this!

The setUp( ) method

- The UnitTest.TestCase class has a setUp( ) method that we can use to assemble our testable components before each test
- Python runs the setUp( ) method before running each method starting with test_
- Any objects created in setUp( ) are available to the tests
- We do this to avoid:
  - Tedious code duplication
  - Tedious set-up repetition or maintenance
- Let's try this out with the Cat class from today's lecture
- Check out test_cat.py

That's it for week 13!

One more week left!

LEGO. It's all about LEGO.

75