Lab 8:

Classes and Exceptions

Object Oriented Programming

- What is it?
  - Programming style where there are *objects* that contain *data* called *attributes* that can be accessed using *methods* called *behaviours*.
  - Java is an example of this kind of programming language
- Python classes
  - Python can be programmed as a scripting language (as we've been doing so far) and as object oriented (which we'll be doing today)
  - Function variables (attributes) and functions (methods) are accessible by outside programs
- Namespacing
  - Your function names, variables, etc. only exist in your class namespace
  - Be careful! Global variables are shared between instances!

Classes vs Programs

- When writing a **program** you are interacting with the **user**
- When writing a **class** you are interacting with a **programmer**

General rule of thumb:

- Do not do command-line handling in a class
  - Unless that's the entire purpose of the class
- Do not provide user-level feedback
  - Error statements should be geared towards programmers
- Classes should handle **one specific task**

Classes

- Python class is defined as:
    - `class ObjectName(BaseClass):`
    - If you don't have a base class you want to inherit, then this will be **object**
- A class will contain:
    - `self`
        - This variable represents the object itself, your ***instance***
        - You can access methods and attributes internally (inside your class)
        - The first argument for **every** method in a class is `self`
        - Like the keyword `this` in Java (but **must** always be used for each method and member variable)
    - `__init__(self, args)`
        - This is the constructor
        - Called once (and only once) upon object instantiation
    - Any other functions and variables you would like

Methods

- Functions that are a member of a class
- Required arguments
    - These are arguments are required by the class/method
    - `def f(self, arg_one, arg_two)`
        - Notice that `self` is a required arguments that is passed by the **interpreter**
        - **It must always be the first argument**
- Optional arguments
    - These are arguments that have default values that are used instead
        - `def f(self, default_str=”some string”):`
        - This function can be called as *`class`*`.f()`
    - You can override the default values
        - e.g. `class.f(default_str=”some other string”)`

Example: Simple Class (hello.py)

```
class HelloWorld(object):

    class_var = “123”    #shared by all instances

    def __init__(self, str=“hello world”)
     self._output = str    #“Private” variable

    def print(self, test=False):
     out_str = self._output if not test else self.class_var
     print(out_str)
```

```
> import hello

> hw = hello.HelloWorld()

> hw.print() → hello world

> hw.print(test=True) → 123
```

Importing and Naming

- To import a class you made, you need to import the **file name** (this is also it’s namespace)
- You can also import a folder of classes!
  - Create an empty file called **__init__.py** in the folder
  - For example, if I had a folder called **lib** with a class file called **util.py**, I could import it by:
    - import lib.util
    - from lib import util
- Private variables do not officially exist in python
  - Private variables are denoted with a single underscore “_”

Example: Locality

```
class Dog:
    tricks = []
    kind = “canine”

    def __init__(self, name):
        self.name = name

    def add_trick(self, trick):
        self.tricks.append(trick)
```

```
> from dog import Dog
> fido = Dog(“Fido”)
> rex =  Dog(“Rex”)
> rex.name → Rex
> fido.name → Fido
> rex.add_trick(“sit”)
> fido.add_trick(“beg”)
```

What do you think rex.tricks will have?

What will fido.tricks have?

Advanced Classes

- We won’t be covering these in the lab, but some neat things you can look at:
  - Comparison overrides
    - https://www.python-course.eu/python3_magic_methods.php
  - Class iterators
    - http://www.diveintopython3.net/iterators.html
  - Decorators
    - https://www.python-course.eu/python3_decorators.php

Exceptions

- Exceptions are a class used for error handling
- You can create your own exceptions using inheritance
  - These are useful for when you are creating your own classes
  - Easier to **raise** an exception than return an error value!

- Raise an exception when your program gets to an unexpected state
- Exceptions can be “caught” and handled using `try/except`

```
try:
    some_code()

except:
    print(“Uh oh… an error occurred … handle it!”, file=sys.stderr)
```

Handling Exceptions

- You can “catch” specific exceptions in a chain

```
try:
     some_code()
except TypeError as e:
     print(“TypeError was thrown: ”, e, file=sys.stderr)
except IndexError as e:
     print(“IndexError was thrown: ”, e, file=sys.stderr)
finally:
     do_something()
```

- **finally** is handy because it always run (whether an exception occurs or not) and before any **return** or **exit** statements

Raising Exceptions

- Raising an exception

```
def input_num(number):
    if number < 0 or number > 10:
        raise InputException(“Input not between 1 and 10”)

    # do some stuff if number is between 0 and 10
    ...
```

- Execution stops when a `raise` is reached

Example: Exception class

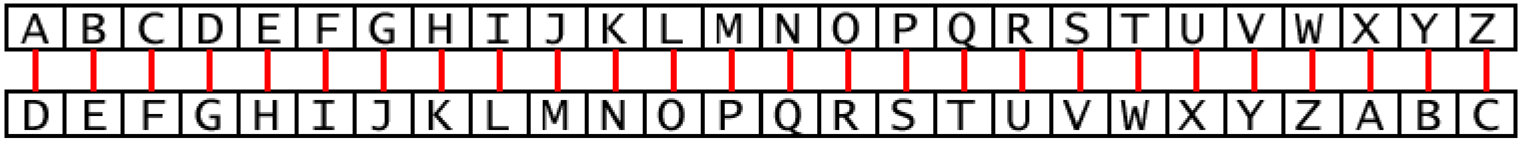
```
import sys
class InputException(Exception):
    pass

try:
        input_num(25)
        raise TypeError()
        print("No problems here!")
except InputException as e:
       print("This except only runs if the error is type InputException: ", e, file=sys.stderr)
except:
       print(“This catches all other errors", file=sys.stderr)
       sys.exit(-1)
finally:
       print(“After everything in the try or excepts have run, then this runs.")

print(“I’m the rest of the program.")
```

In-Lab Activity: Caesar Cipher

Your task is to create a cipher class called **CaesarCypher** (**caesar.py**). A caesar cipher is one of the simplest ciphers to create. You provide a shift key that shifts the letter mapper as shown below:


A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
D E F G H I J K L M N O P Q R S T U V W X Y Z A B C

Alphabet shifted by 3 spaces.

Plaintext: THEQUICKBROWNFOXJUMPSOVERTHELAZYDOG
Ciphertext: WKHTXLFNEURZQIRAMXPSVRYHUWKHODCBGRJ

In-Lab Activity: Caesar Cipher

Your class should have the following functions:

- __init__(self, key) ➝ Constructor that takes in a integer shift key
- encrypted_text = encrypt(self, text) ➝ encrypt text
- original_text = decrypt(self, encrypted_text) ➝ decrypt text

You can assume that only the letters A-Z are in the text.
 - Optional: Validate the input and throw an ValueError if the input contains characters that are not A-Z
- https://docs.python.org/3/library/exceptions.html#ValueError

You are provided a tester script **run_caesar.py** to help test your class.