ENDG 233 – Programming with Data

Modules and Classes

Week 11: Nov. 15 - 21

Schedule for Week 11

UNIVERSITY OF CALGARY

- Review modules and classes
- Make sure required modules are installed properly
- Work on assignment #3

- Wednesday:
  - Continue working on assignment #3
  - Introduce final project

- Next week: Term Test #2 on Tuesday!

Review: Modules

- A programmer may find themselves writing the same function over and over again in multiple scripts
- A solution is to use a **module**, which is a file containing Python code that can be imported and used by scripts or other modules
- To **import** a module means to execute the code contained by the module, and make the definitions within that module available for use by the importing program

Review: Modules

- A module's filename should end with ".py“. Otherwise, the interpreter will not be able to import the module
- The module_name item should match the filename of the module, but without the .py extension
- Ex: If a programmer wants to import a module whose filename is HTTPServer.py, the import statement *import HTTPServer* would be used
- **Note**: import statements should be placed at the top of the script

Review: Built-in Modules

- You have already used multiple built-in modules in this course
  - math, string, etc.
- Modules you will need for the rest of this course:
  - matplotlib, numpy

Review: Modules and venv

- You can install modules within your virtual environment using pip
- Command: *pip install module_name*

  or

  python –m pip install module_name
- To run your code with the installed modules, you need to either:
  - Run the code from the terminal (python command)
  - Select the venv python interpreter

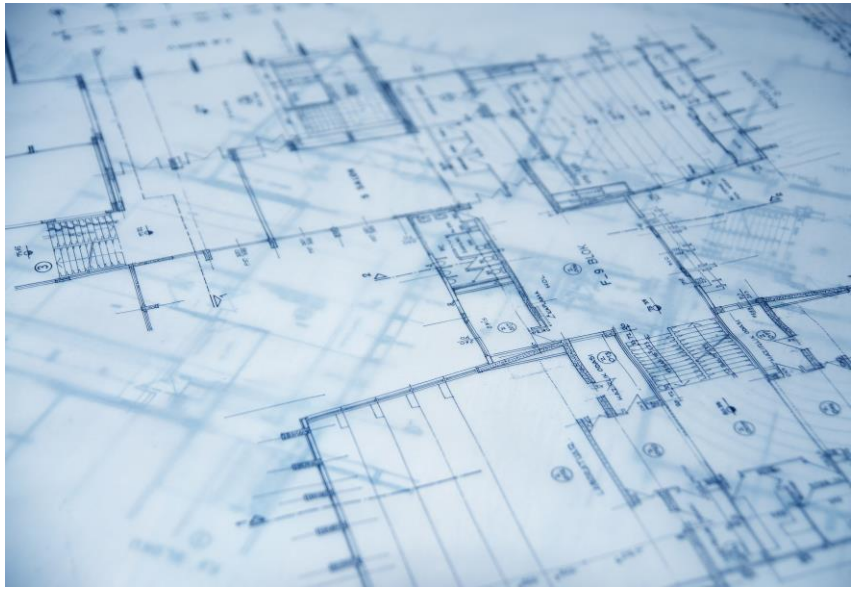
Classes and objects

- Classes are templates for creating objects similar to how you create a variable of different types
- Classes provide the attributes (variables) and functionality (methods) of the object to be created from the template

Classes and objects: Blueprint and houses

Class ??

Information of what constitutes a house, where the walls will be built, total space size, room placement, etc…



Objects ??

Created from template but each one is unique in color, furnishing, wall paint, window style, door colors and material, etc…

Classes and objects: Factory and Cars

Class ??

Information of car parts, construction, and build

Objects ??

Created from template but each one is unique in color, furnishing, engine capacity, etc….

Classes and objects: Table header and Rows

Class ??

Header represents data provided in table and possible relationship and derived information

Objects ??

Each row represents the values for the header for a unique case

City	Max Temperature	Min Temperature	Avg Temperature = (Max+Min)/2
Calgary	-30	-40	-35
Edmonton	-10	-20	-15
….	…	…	…
….	…	…	…
Vancouver	-2	-10	-6

Review: Objects

- The physical world is made up of material items like wood, metal, plastic, fabric, etc.
- To keep the world understandable, people think in terms of higher-level objects, like chairs, tables, and TV's
- Those objects are groupings of the lower-level items
- Likewise, a program is made up of lower-level items like variables and functions

Review: Objects

- To keep programs understandable, programmers often deal with higher-level groupings of those items, known as objects
- In programming, an object is a grouping of data (variables) and operations that can be performed on that data (functions or methods)
- Creating a program as a collection of objects can lead to a more understandable, manageable, and properly-executing program

Review: Classes

- Multiple variables are frequently closely related and should thus be treated as one variable with multiple parts
- For example, two variables called hours and minutes might be grouped together in a single variable called time
- The class keyword can be used to create a user-defined type of object containing groups of related variables and functions

Review: Class Definitions

- An **instantiation operation** is performed by "calling" the class, using parentheses like a function call
- An instantiation operation creates an **instance**, which is an individual object of the given class
- An instantiation operation automatically calls the **__init__** method defined in the class definition
- The __init__ method, commonly known as a **constructor**, is responsible for setting up the initial state of the new instance

Review: Class Definitions

- The __init__ method has a single parameter "self", that automatically references the instance being created
- The self parameter is reference to the current instance of class and is used to access the variables that belongs to the class.
- Attributes can be accessed using the **attribute reference operator** "." (sometimes called the member operator or dot notation)

Review: Class Example #1

UNIVERSITY OF CALGARY

```
class Time:
    """ A class that represents a time of day """
    def __init__(self):
        self.hours = 0
        self.minutes = 0



my_time = Time()
my_time.hours = 7
my_time.minutes = 15
print('{} hours'.format(my_time.hours), end=' ')
print('and {} minutes'.format(my_time.minutes))
```

Review: Class functions

- A function defined within a class is known as an **instance method**
- An instance method can be referenced using dot notation

Review: Class Example #2

UNIVERSITY OF CALGARY

```
class Time:
    """ A class that represents a time of day """
    def __init__(self):
        self.hours = 0
        self.minutes = 0

    def print_time(self):
        print('{} hours'.format(self.hours), end=' ')
        print('and {} minutes'.format(self.minutes))

my_time = Time()
my_time.hours = 7
my_time.minutes = 15
my_time.print_time()
```

Review: Class constructors

- A class instance is commonly initialized to a specific state
- The __init__ method constructor can be customized with additional parameters, as shown on the next slide

Review: Class Example #3

UNIVERSITY OF CALGARY

```
class Time:
    """ A class that represents a time of day """
    def __init__(self, hours, mins):
        self.hours = hours
        self.minutes = mins


hours = input()
mins = input()

my_time = Time(hours, mins)

print('{} hours'.format(my_time.hours), end=' ')
print('and {} minutes'.format(my_time.minutes))
```

Tutorial 11.1: Calculator class

- Write a class called Calculator that emulates basic functions of a calculator: add, subtract, multiply, divide, and clear
- The class has one attribute called value for the calculator's current value

Tutorial 11.1: Calculator class

UNIVERSITY OF CALGARY

- Implement the following methods as listed below:
  - Default constructor method to set the attribute to 0.0
  - add(self, val) - add the parameter to the attribute
  - subtract(self, val) - subtract the parameter from the attribute
  - multiply(self, val) - multiply the attribute by the parameter
  - divide(self, val)- divide the attribute by the parameter
  - clear(self) - set the attribute to 0.0
  - get_value(self) - return the attribute

Tutorial 11.1: Calculator class

UNIVERSITY OF CALGARY

- Given two float input values num1 and num2, the program outputs the following values:
  - The initial value of the instance attribute, *value*
  - The value after adding num1
  - The value after multiplying by 3
  - The value after subtracting num2
  - The value after dividing by 2
  - The value after calling the clear() method

- **Note**: Only add the class definition, do not change the existing code!

Tutorial 11.1: Calculator class solution

```
if __name__ == "__main__":
        calc = Calculator()
        num1 = float(input())
        num2 = float(input())
        print(f'{calc.get_value():.1f}’)          # 1. The initial value
        calc.add(num1)              # 2. The value after adding num1
        print(f'{calc.get_value():.1f}')
        calc.multiply(3)            # 3. The value after multiplying by 3
        print(f'{calc.get_value():.1f}')
```

Tutorial 11.1: Calculator class solution

UNIVERSITY OF CALGARY

```
calc.subtract(num2)      # 4. The value after subtracting num2
print(f'{calc.get_value():.1f}')
calc.divide(2)           # 5. The value after dividing by 2
print(f'{calc.get_value():.1f}')
calc.clear()             # 6. The value after clearing
print(f'{calc.get_value():.1f}')
```

Tutorial 11.1: Calculator class solution

```
class Calculator:
    def __init__(self):
        self.value = 0.0
    def add(self, val):
        self.value += val
    def subtract(self, val):
        self.value -= val
    def multiply(self, val):
        self.value *= val
    def divide(self, val):
        self.value /= val
    def clear(self):
        self.value = 0.0
    def get_value(self):
        return self.value
```

Tutorial 11.3: Car value (classes)

- Task : Complete the Car class by creating an attribute purchase_price (type int) and the method print_info() that outputs the car's information.
- If the input is:
  - 2011
  - 18000
  - 2018
- Output is:
  - Car's information:
  - Model year: 2011
  - Purchase price: 18000
  - Current value: 5770

Tutorial 11.3: Car value (classes)

UNIVERSITY OF CALGARY

```
class Car:
    def __init__(self):
        self.model_year = "none"
        self.purchase_price = 0
        self.current_value = 0

    def calc_current_value(self, current_year):
        depreciation_rate = 0.15
        car_age = current_year - self.model_year
        self.current_value = round(self.purchase_price * (1 - depreciation_rate) ** car_age)
```

Tutorial 11.3: Car value (classes)

UNIVERSITY OF CALGARY

```
def print_info(self):
        print('Car\'s information:')
        print(f'   Model year: {self.model_year}')
        print(f'   Purchase price: {self.purchase_price}')
        print(f'   Current value: {self.current_value}')
```

Tutorial 11.3: Car value (classes)

```
if __name__ == "__main__":
    year = int(input())
    price = int(input())
    current_year = int(input())

    my_car = Car()        # car object
    my_car.model_year = year
    my_car.purchase_price = price
    my_car.calc_current_value(current_year)
    my_car.print_info()
```

Tutorial 11.4: Triangle area comparison (classes)

UNIVERSITY OF CALGARY

```
class Triangle:
    def __init__(self):
        self.base = 0
        self.height = 0

    def set_base(self, user_base):
        self.base = user_base

    def set_height(self, user_height):
        self.height = user_height
```

Tutorial 11.4: Triangle area comparison (classes)

UNIVERSITY OF CALGARY

```
def get_area(self):
        area = 0.5 * self.base * self.height
        return area

    def print_info(self):
        print(f'Base: {self.base:.2f}')
        print(f'Height: {self.height:.2f}')
        print(f'Area: {self.get_area():.2f}')
```

Tutorial 11.4: Triangle area comparison (classes)

UNIVERSITY OF CALGARY

```
if __name__ == "__main__":
    triangle1 = Triangle()
    triangle2 = Triangle()

    # Read and set base and height for triangle1 (use set_base() and set_height())
    user_base = float(input())
    user_height = float(input())
    triangle1.set_base(user_base)
    triangle1.set_height(user_height)
```

Tutorial 11.4: Triangle area comparison (classes)

UNIVERSITY OF CALGARY

```
# Read and set base and height for triangle2 (use set_base() and set_height())
user_base = float(input())
user_height = float(input())
triangle2.set_base(user_base)
triangle2.set_height(user_height)

# Determine larger triangle (use get_area())
print('Triangle with larger area:')
if triangle1.get_area() > triangle2.get_area():
    # Output larger triangle's info (use print_info())
    triangle1.print_info()
else:
    # Output larger triangle's info (use print_info())
    triangle2.print_info()
```