

# CSE231 - Lab 10

Class Justification

# Program Development

As we get closer and closer to knowing everything about our first programming language, the more we want to look towards applying our knowledge to real-world situations.

Classes are extremely useful to organize and abstract away code. When we use programming to tackle a problem, we generally want to dedicate some portions of our code to something, and leave other portions to something else. Like functions, breaking our code into sections with classes can provide a better “mental structure” of what’s happening during the breakdown of a problem, and can eliminate time spent debugging among other things.

We want you to begin thinking about how classes work. For right now, we’re going to get some practice *using another, custom class* and later transition to making our own next week.

# Problem Solving Principles

1. Understand the problem; Can you think of a picture/diagram that might help you understand the problem?
2. Devise a plan; Try creating a pseudocode algorithm for how you want to tackle the problem! Perhaps try to solve a simpler version of the problem!
3. Carry out the plan
4. Review/Extend

You might think that all of this is fairly unnecessary. But as you deal with problems that scale in complexity *outside of this class*, these principles can be extremely helpful. Creating classes give you the opportunity to simulate a diagrammatic way of thinking in code.

# What is a class?

We've secretly been using classes this entire time. Dictionaries, tuples, strings, ints, floats -- all of these are classes.

```
print(type("Hello World!"))    # <class 'str'>
```

It's important to note here that “types” as we've been referring to them traditionally are synonymous with classes. The string we made above is an *instance of a class, called 'str', that holds a value: "Hello World!"*

Strings, lists, dictionaries, etc. are all classes that act as “templates” for the kind of data you want to store. They're helpful to us because they work in *general* situations. You can store *anything* inside a list or tuple, as an example -- we want to apply that same principle to make classes that can manage *anything* related to a particular problem.

# General-Case Programming

```
def general_add(a, b):  
    if type(a) == str:  
        a = int(a)  
  
    if type(b) == str:  
        b = int(b)  
  
    return a + b
```

We could, for example, create a function that adds two numbers in both int *and* string form. We cover more options this way, but at what cost?

# Problems With General-Case Programming

```
def general_add(a, b):
```

```
    if type(a) == str:
```

```
        a = int(a)
```

```
    if type(b) == str:
```

```
        b = int(b)
```

```
    return a + b
```

```
general_add("2uh-oh", 3)    # Error
```

# Problems With General-Case Programming (cont.)

The point I'm trying to make here is that programming *as general as possible* sometimes isn't the best option. It depends on the problem you're trying to solve. The problem we just ran into is a problem of generality in itself -- the ``str`` class isn't restricted to just hold number values.

If, for example, *you know for sure* that a string like "2uh-oh" would never be input to the function, then yeah, your function might be good for your use-case.

When tackling a problem, you'll have to make the call on *how general you want to get*. Classes offer *the most fundamental way of general-case programming*. You'll commonly hear this referred to as **Object-Oriented Programming**. It's extremely powerful, sometimes overused, and can be extremely confusing at times.

# Class Behavior

The reason classes are so general in nature is because the programmer decides *how the class interacts with the language*. Let's take the list and dictionary for example.

Lists have `.sort()`, `.pop()`, `.append()`...

Dictionaries have `.keys()`, `.values()`, `.items()`...

Method functions are one of the ways we can dictate how our class operates. The concept of a list is fundamentally different from dictionaries, and so the Python developers created method functions specifically dedicated to the class.



Demo Time