Student Guide

```
class Student:
  def __init__(self, name, grade, major):
    self.name = name
    self.grade = grade
    self.major = major

  def submit_graduation_form(self):
    # TODO: submit generic university forms
    pass

class CS_Major(Student):
  def __init__(self, name, grade, major_track):
    super().__init__(name, grade, "CS")
    self.major_track = major_track

  def submit_graduation_form(self):
    # TODO: also submit CS specific forms
    super().submit_graduation_form()

you = CS_Major("your name", "senior",
  "cybersecurity")
```

We are going to be talking about *inheritance*. Here are some defintions:

- **Superclass**: A generalized class; A general class fromwhich a more specialized class inherits (e.g. `Student` is a *superclass* of `CS_Major`).
- **Subclass**: More specialzied class; A class that inherits variables and methods from a superclass but may also add instance variables, add methods, or redefine methods (e.g. `CS_Major` is a *subclass* of `Student`)

And, finally:

- **Inheritance**: Is this idea that a specialized subclass can still make use of instance variables and methods of its more general superclass. (e.g. `CS_Major` *inherits from* `Student`).

A **subclass** *inherits* from a **superclass**. Every subclass is an instance of a superclass.

`super()`

Just as `self` provides access to the current instance at hand, `super()` provides access to the instance we are inheriting from. Note that `super()` is a **function** -- parentheses are required.

In every class (and in our example above), the subclass `__init__()` **must** call the superclass `super().__init__()`! This allows the child instance to inherit all of the instance variables of the parent instance.

Additionally, whenever the superclass defines a function `f`, we can choose to call `super().f()` when defining the current `f`. (See `submit_graduation_form` above). If we do this, we say we *inherit* the parent's method `f`. If we do not do this, we *override* the parent's method `f`.

`isinstance`

`isinstance(object, class)` checks whether `object` is an instance (or subclass!) of `class`.

When `object` is **directly** an instance of `class`, this has the same effect as `type(object, class)`. For example, we could write `isinstance("hello", str)` instead of `type("hello") == str`.

The key difference between `isinstance` and `type` is that `isinstance` supports inheritance. For example:

```
class Vehicle:
    pass

class TrainEngine(Vehicle):
    pass

thomas = TrainEngine()
```

observe that

```
isinstance(thomas, Vehicle) # returns True
type(thomas) == Vehicle     # returns False
```

File Locations for Credit

These are the paths we will be grading on (spelling is important -- make sure to capitalize exactly as shown and **NO** spaces in `Lab13`):

- `Laboratory/Lab13/Others.py`
- `Laboratory/Lab13/Shape.py`
- `Laboratory/Lab13/Canvas.py`