2.4 Inheritance: Introduction and Methods

INHERITANCE:

- Relationship between two classes
- Making a base class (has methods that are shared in common / can be factored put)
- Making subclasses (have methods that are more specific & inherit the common methods from base class)

"If class B is a subclass of class A, then A is a superclass of B"

TERMINOLOGY:

- Base class is a.k.a : superclass , parent class
- Subclass is a.k.a: derived class, child class

ABSTRACT CLASS:

- Has at least one abstract method: *shouldn't be instantiated & raises NotImplemented error*

```
class Employee:
    """An employee of a company.

    This is an abstract class. Only subclasses should be instantiated.
    """
    def get_monthly_payment(self) -> float:
        """Return the amount that this Employee should be paid in one month.

        Round the amount to the nearest cent.
        """
        raise NotImplementedError

    def pay(self, pay_date: date) -> None:
        """Pay this Employee on the given date and record the payment.

        (Assume this is called once per month.)
        """
        payment = self.get_monthly_payment()
        print(f'An employee was paid {payment} on {date}.')
```

- The abstract method is redefined in the subclasses

```
class SalariedEmployee(Employee):
    def get_monthly_payment(self) -> float:
        # Assuming an annual salary of 60,000
        return round(60000.0 / 12.0, 2)

class HourlyEmployee(Employee):
    def get_monthly_payment(self) -> float:
        # Assuming a 160-hour work month and a $20/hour wage.
        return round(160.0 * 20.0, 2)

>>> fred = SalariedEmployee()
>>> fred.get_monthly_payment()
5000.0
>>> jerry = HourlyEmployee()
>>> jerry.get_monthly_payment()
3200.0
```

RULE TO REMEMBER:

“ type(self) determines which class python first looks in for the method"