# Object Oriented Programming 101: Classes

The idea of object oriented programming (OOP) is that it makes it easier for you (and for others) to design, implement, test, debug and update your code. A key component to learning how to write object oriented code in Python is the use of Classes.

So far, we've actually used various *existing* classes like `int, float, s tr, list, tuple, dict` and etc. With this lesson, we will cover how to create your own custom classes.

1. **Creating a class:**
   - Python's style guide is to capitalize each word ('BankAccount').
   - The **__init__** method is used to initialize various attributes of the object (e.g. a 'Car' object would most likely initialize its make, model, year, etc. __init__ will automatically execute whenever you create a new instance of a Class.
   - Bonus: You can make an attribute 'private' by adding two underscores as a prefix (see. `self.__user_id` below). This makes the attribute inaccessible to users of the Class.

```
class Account:
    """Account class for maintaining a bank account"""

    def __init__(self, name, balance):
        """Initialize an Account object"""
        self.name = name
        self.balance = balance
        self.__user_id = random.randint(pow(10, 5), pow(10, 8))

    def deposit(self, amount):
        """Deposit money to the account"""
        self.balance += amount

    ...more code (withdraw, log_transactions, etc)
```

2. **Using a class**: An instance of a class turns the object into a 'real' thing.

```
# Create an instance of a class.
james_account = Account('James Gosling', 20000000)

# Let's deposit some money in there
james_account.deposit(500)

# Let's see James's balance
print(james_account.balance)
```

3. **Modularity**: Classes allow you to modularize your code by using import statements.

```
import Account

guido_account = Account('Guido van Rossum', 10000000)
tim_account = Account('Tim Berners-Lee', 12000000)

...other pieces of code that builds upon the imported Class
```

4. **Inheritance**: Classes can inherit other classes. The inheritor is called a "child" class. When a child class is created, it inherits all the attributes / functions / etc of its parent.
   - You can add new functions that complement the parent's class, or even override the parent class's functions/attributes.
   - **@property:** In short, @property makes attributes a read-only property, which allows you to retrieve the attribute like a property (ie. self.name). They are a form of a **decorator function**.

```
class SavingsAccount(Account):
    """Initialize a SavingsAccount object"""
    @property
    def interest_rate(self):
        """Return the interest_rate of the account"""
        return 0.00299

    def calc_interest_gain(self):
        """Calculate annual interest rate"""
        interest = self.interest_rate
        return self.balance * interest

yukihiro_account = SavingsAccount('Yukihiro Matsumoto',
15000000)
print(yukihiro_account.interest_rate) ## 0.00299
print(yukihiro_account.calc_interest_gain()) ## 44850.0
```