Caesar Cipher

Making Game with Python (1)

Zhihong (John) Zeng & Andrew Zeng

Agenda

- Python classes and objects
- Caesar Cipher

Python Classes and Objects

Objects

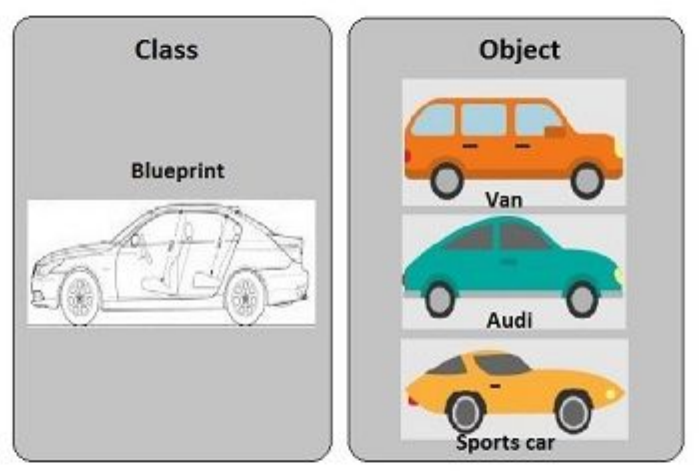
- Python supports many different kinds of data
  - 1234, 3.14159, ‘Hello’, [1, 2, 5, 7], {‘CA’: ‘California’, ‘MA’: ‘Massachusetts’}
- Each is an object, and every object has:
  - A type
  - An internal data representation
  - A set of procedures for interaction with the object
- An object is an instance of a type
  - 1234 is an instance of an int
  - ‘Hello’ is an instance of a string

Object Oriented Programming (OOP)

- EVERYTHING IN PYTHON IS AN OBJECT
- Can create new objects of some type
- Can manipulate objects

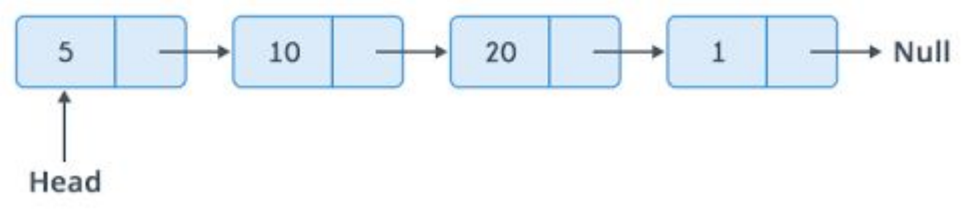
What are objects?

- Objects are a data abstraction that captures …
    - An internal representation through data attributes
    - An interface for interacting with object through functions/methods
- Example



Example in python: L = [5, 10, 20]

- How are lists represented internally? Linked list of cells



- How to manipulate lists?
  - L[i], L[i:j]
  - len(), min()
  - L.append()

User-defined Classes and Objects

- Distinction between creating a class and using the class to create an object
- Creating a class involves:
  - Defining the class name
  - Defining data attributes/methods
- Using the class involves:
  - Creating new instances of objects
  - Doing operations on the instances

Define your own class

```
class Person:

    def __init__(self, name, age):
        self.name = name
        self.age = age

    def hello(self, greeting):
        print(greeting + ' ' + self.name)

x = Person('John', 10)
x.hello('Hello')
print(x.age)
x.age = 20
print(x.age)
```

Keyword class

Initialization

Method to interact

Create an object
Interact with object
Access to object attribute
Change object attribute

The self parameter

The self is the reference to the current instance of the class, and is used to access the instance data and methods

```
class Person:

    def __init__(self, name, age):
        self.name = name
        self.age = age

    def hello(self, greeting):
        print(greeting + ‘  ‘ + self.name)

x = Person('John', 10)
x.hello(‘Hello’)
```

traditional function

```
def hello(name, greeting):
    print(greeting + ‘  ‘ + name)


hello(‘John’, ‘Hello’)
```

Class Inheritance

- Inheritance allows us to define a class that inherits all the methods and attribute from another class
- Parent class is the class being inherited from, also called base class
- Child class is the class that inherits from another class, also called derived class

```
class Person:

    def __init__(self, name, age):
        self.name = name
        self.age = age

    def hello(self, greeting):
        print(greeting + ' ' + self.name)

x = Person('John', 10)
x.hello('Hello')
```

```
class Student(Person):

    def __init__(self, name, age, grade):
        super().__init__(name, age)
        self.grade = grade

    def print_grade(self):
        print(self.name + '  is at Grade  ' + str(self.grade))

y = Student('Peter', 10, 4)
y.hello('Hello')
y.print_grade()
```

Create object

Same method

Additional method

Polymorphism from Inheritance

- Polymorphism is the ability to take on many forms
- Parent class and child class may take different implementation for the same method

```
class Person:

    def __init__(self, name, age):
        self.name = name
        self.age = age

    def hello(self, greeting):
        print(greeting + ' ' + self.name)

x = Person('John', 10)
x.hello('Hello')
```

```
class Professor(Person):

    def __init__(self, name, age):
        super().__init__(name, age)

    def hello(self, greeting):
        print(greeting + ' Prof ' + self.name)

p = Professor('Gates', 10)
p.hello('Hello')
```

Override method

Different result

Power of OOP

- Encapsulation:
  - wrapping the properties together in a single unit,
  - efficient to organize and maintain the code
  - Python use double underscore (__) to indicate the private properties
- Inheritance:
  - one class inherits properties from another class
  - code reusability
- Polymorphism:
  - have many implementation for same method
  - Code flexibility

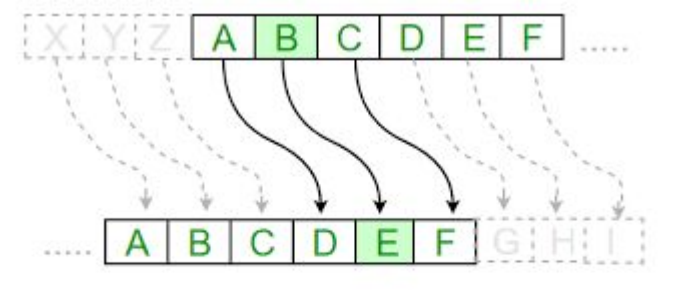
Caesar Cipher

Caesar Cipher

Ever want to pass secret messages to your friends? Well, here is your chance! But first, here is some vocabulary:

- Encryption - the process of obscuring or encoding messages to make them unreadable
- Decryption - making encrypted messages readable again by decoding them
- Cipher - algorithm for performing encryption and decryption
- Plaintext - the original message
- Ciphertext - the encrypted message

Caesar Cipher (cont)



- The idea of the Caesar Cipher is to pick an integer and shift every letter of your message by that integer.
  - Suppose the shift is k. Then, all instances of the i letter of the alphabet that appear in the plaintext should become the (i + k)th letter of the alphabet in the ciphertext. You will need to be careful with the case in which i + k > 26 (the length of the alphabet).
- We will treat uppercase and lowercase letters individually, so that uppercase letters are always mapped to an uppercase letter, and lowercase letters are always mapped to a lowercase letter.
- Punctuation and spaces should be retained and not changed.

Classes and Inheritance

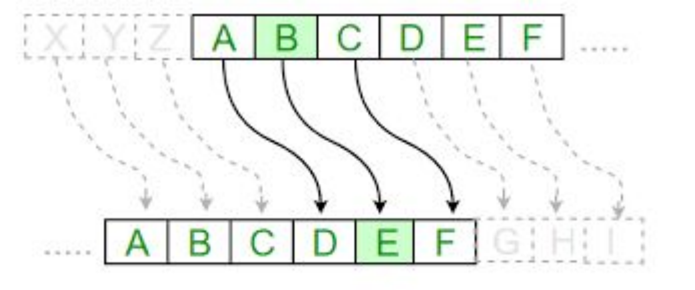
- We will have a Message class with two subclasses PlaintextMessage and CiphertextMessage .
- Message contains methods that could be used to apply a cipher to a string, either to encrypt or to decrypt a message (since for Caesar codes this is the same action).
- PlaintextMessage has methods to encode a string using a specified shift value; our class will always create an encoded version of the message, and will have methods for changing the encoding.
- CiphertextMessage contains a method used to decode a string.

Message Class

```
import string
class Message(object):
    def __init__(self, text):
        self.message_text = text

    def build_shift_dict(self, shift):
        def shift_func(letter_list, shift):
            letter_list_rotate = letter_list[shift:] + letter_list[:shift]
            ans = {}
            for k, v in zip(letter_list, letter_list_rotate):
                ans[k] = v
            return ans

        assert shift >= 0 and shift < 26, 'Error: shift should be in [0, 26), but is {}'.format(shift)
        self.shift_dict = shift_func(list(string.ascii_lowercase), shift)
        self.shift_dict.update(shift_func(list(string.ascii_uppercase), shift))
```



Message Class (cont)

```
class Message(object):
    ....

    def apply_shift(self, shift):
        self.build_shift_dict(shift)
        ans = "
        for x in self.message_text:
            if x in string.ascii_letters:
                ans += self.shift_dict[x]
            else:
                ans += x
        return ans
```

PlaintextMessage Class

```
class PlaintextMessage(Message):
    def __init__(self, text, shift):
        super().__init__(text)
        self.shift = shift

    def get_message_text_encrypted(self):
        self.message_text_encrypted = self.apply_shift(self.shift)
        return self.message_text_encrypted
```

CiphertextMessage Class

```
from load_check_words import load_words, get_story_string, get_num_valid_words

WORDLIST_FILENAME = 'words.txt'

class CiphertextMessage(Message):
    def __init__(self, text):
        super().__init__(text)
        self.valid_words = load_words(WORDLIST_FILENAME)
```

CiphertextMessage Class (cont)

```
class CiphertextMessage(Message):
    ...
    def decrypt_message(self):
        max_valid_words = 0
        best_shift = 0
        decrypted_message_text = "
        for shift in range(26):
            text = self.apply_shift(shift)
            num_valid_words = get_num_valid_words(text, self.valid_words)

            if max_valid_words < num_valid_words:
                max_valid_words = num_valid_words
                best_shift = shift
                decrypted_message_text = text

        return (best_shift, decrypted_message_text)
```

Test

```
if __name__ == '__main__':
    plaintext = PlaintextMessage('hello', 2)
    print('Expected Output: jgnnq')
    print('Actual Output:', plaintext.get_message_text_encrypted())

    ciphertext = CiphertextMessage('jgnnq')
    print('Expected Output:', (24, 'hello'))
    print('Actual Output:', ciphertext.decrypt_message())

     # test to decrypt story text
    story = get_story_string()
    print('\nEncrypted story:\n', story)

    cipher_story = CiphertextMessage(story)
    print('\nDecrypted story: \n', cipher_story.decrypt_message())
```

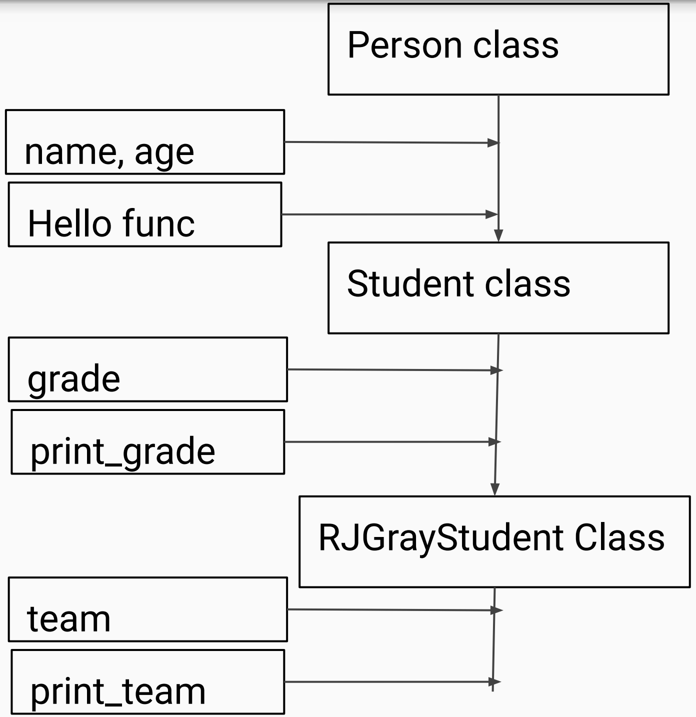
Python great child class

```
class RJGrayStudent(Student):

    def __init__(self, name, age, grade, team):
        super().__init__(name, age, grade)
        self.team = team

    def print_team(self):
        print(self.name + ' is at Team ' + self.team)

z = RJGrayStudent('Adam', 12, 7, 'Blue')
z.hello('Hello')
z.print_grade()
z.print_team()
```


Person class
name, age
Hello func
Student class
grade
print_grade
RJGrayStudent Class
team
print_team