What is Object Oriented Programming?

Object-Oriented Programming in a nutshell

Till now you have dealt with functions manipulating data inside them which is also called the *procedure-oriented* way of programming. But there is another more powerful way of organizing your program which gives you the flexibility to deal with data and functionality and wrap it inside something called an object. Since Python is a multi-paradigm language, it supports object-oriented programming (OOP).

Class and objects are two important aspects in OOP. Class creates a new *type* altogether whereas object is an instance of an a class. These can have their own attributes i.e. characteristics and methods i.e. actions.

Example of OOP

For instance, John is an object of the class human with attributes like name, age and methods like speaking, eating etc.

Properties of OOP

Like any other language, in Python, the concept of OOP follows some basic principles:

Property	Description
Inheritance	A process of using details from a new class without modifying existing class
Encapsulation	Hiding the private details of a class from other objects

Polymorphism	A concept of using common operation in different ways for different data input

Creating your Own Class

Now time to create your own classes. Lets go through the code below to see what a class looks like and how to make one.

```
class Person(object):

    # initialize
    def __init__(self, name, title, gender):
        self.name = name
        self.title = title
        self.gender = gender

    # display full name
    def display_name(self):
        return self.name + ' ' + self.title

    # display gender
    def isgender(self):
        return self.gender

    # change first name
    def change_first_name(self, new_name):
        self.name = new_name

    # change title
    def change_title(self, new_title):
        self.title = new_title

# object name "p"
p = Person('Rita', 'Roy', 'female')

# display full name
print(p.display_name())
print('='*50)

# display gender
print(p.isgender())
print('='*50)

# change first name
p.change_first_name('Amrita')

# change title
p.change_title('Ganguly')

# print full name
print(p.display_name())
```

Output

```
'Rita Roy'
'================================================='
'female'
'================================================='
'Amrita Ganguly'
```

Here,

- A new class begins with the class keyword followed by a name Person
- The object part in parentheses specifies the parent class that you are inheriting from.
- There is an init() method defined with `def` which instantiates the object for the class. This method takes the self as an argument always. self is nothing but the object itself.
- The attributes name, title and gender are instance attributes since for every instance they will be different. Remember that only for class attributes it/they will be the same for every instance. We initialize the first name, title, and gender with `self.name=name`, `self.title=title` and `self.gender=gender`
- `display_full_name()` is a method which returns the full name of the object
- `isgender()` full name is a method that returns the gender of the object.
- The method `change_first_name()` takes in a new_name as argument and modifies the original name.
- The method `change_title()` takes in a new_title as argument and modifies the original title.
- `p` is an instance of the object Person class with its arguments as `Rita`, `Roy` and `female`.
- Now, if we do `p.displa_first_name()`, we can see that the full name is displayed i.e. `'Rita Roy'`.
- Now change the first name to `'Amrita'` with `.change_first_name()` and change the title to `'Ganguly'` with `.change_title()`
- If you see the new fill name with `p.display_name()` then you can see that `'Rita Roy'` has changed to `'Amrita Ganguly'`

Message Reading

The first thing you have to do is to write a function that reads the contents of the files that we have.

Instructions:

- The path for file has been stored in a variable `file_path`
- Write a function `"read_file()"` that :
    - Takes `'path'` as a parameter.
    - Opens the file associated with the `'path'` in the read-only mode `('r')` and store it in a variable `file`.
    - Reads the content(first line) of the file and stores it in a variable called `'sentence'`
    - Closes the file
    - Returns `'sentence'`

Parameters :

parameter	dtype	Argument Type	default value	description
path	string	compulsory		path of the file location

Returns:

returns	dtype	description
sentence	string	Sentence in the file

- Call the function `"read_file()"` with `'file_path'` as input parameter and store the result in a variable called `'sample_message'`

```
##File path for the file
file_path



#Code starts here
def read_file(path):
    file = open(file_path,'r')
    lines = []
    for line in file.readlines():
        lines.append(line)
    sentence = lines[0]
    file.close()
    return sentence

sample_message = read_file(file_path)
```

Message Fusion

In this task, you have to make use of messages of two different files. In the two files, we have one number each.

You have to apply a certain operation to extract our message.

Instructions:

- Path for the two files that you will require for this task has been stored in variables `file_path_1` and `file_path_2`
- Call the function `read_file()` written in the previous task for `file_path_1` & `file_path_2` and store their message sentences in variables `message_1` and `message_2` respectively.
- Print `message_1` and `message_2` to see what they contain.
- Write a function `fuse_msg()` that :
    - Takes `message_a` and `message_b` as parameters

    - Implements integer(floor) division of `message_b` over `message_a`(Don't forget to make messages as int before applying floor division) and stores the quotient in a variable called `quotient`
    - Returns `quotient` in string format.
- [Note you can convert any variable 'a' to string using str(a)]

Parameters :

parameter	dtype	Argument Type	default value	description
message_a	string	compulsory		message from the first text file
message_b	string	compulsory		message from the second text file

Returns:

returns	dtype	description
quotient	string	quotient of the integer division

- Call the function `fuse_msg()` with `message_1` & `message_2` and store the result of it in a variable called `secret_msg_1`

```
message_1 = read_file(file_path_1)
message_2 = read_file(file_path_2)

print(message_1)
print(message_2)

def fuse_msg(message_a,message_b):
   quotient = int(message_b)//int(message_a)
   return str(quotient)

secret_msg_1 = fuse_msg(message_1,message_2)
```

In this task, you have to substitute the message of the file for a secret message.

Instructions:

- Path for the file that you will require for this task has been stored in variables `file_path_3`
- Call the function `read_file()` for `file_path_3` and store its message sentences in variables `message_3`
- Print `message_3` to see what it contains.
- Write a function `substitute_msg()` that :

- Takes `message_c` as a parameter

Creates a new variable `'sub'` and in it stores

```
'Army General' if message_c is 'Red'
'Data Scientist' if message_c is 'Green'
'Marine Biologist' if message_c is 'Blue'
```

-
- Returns `'sub'`

Parameters :

parameter	dtype	Argument Type	default value	description
message_c	string	compulsory		message from the text file

Returns:

returns	dtype	description
sub	string	word according to the condition given

- Call the function `"substitute_msg()"` with `'message_3'` and store the result of it in a variable called `'secret_msg_2'`

```
#Code starts here
message_3 = read_file(file_path_3)
print(message_3)

def substitute_msg(message_c):
    if message_c == 'Red':
        sub = "Army General"
    elif message_c == 'Green':
        sub = "Data Scientist"
    elif message_c == 'Blue':
        sub = "Maine Biologist"
    else:
        pass
    return sub

secret_msg_2 = substitute_msg(message_3)
```

Message Comparision

In this task, you have to make use of messages from two different files. You have to compare the two messages and take only those words that appear in first message but not in second message.

Instructions:

- Path for the two files that you will require for this task has been stored in variables `file_path_4` and `file_path_5`
- Call the function `read_file()` for `file_path_4` & `file_path_5` and store their message sentences in variables `message_4` and `message_5` respectively
- Print `message_4` and `message_5` to see what they contain.
- Write a function `compare_msg()` that :
    - Takes `message_d` and `message_e` as parameters
- Breaks down the sentences in `message_d` & `message_e` into words using `split()` function and stores them in `a_list` & `b_list` respectively
    - Stores all the words that are there in `a_list` but not in `b_list` in a new list called `c_list`
    - Combines the words of `c_list` back to a sentence using `join()` and stores it in a variable called `final_msg` and returns it

---

`Example of join function :`

```
Word_List=['I', 'love', 'data']
Sentence= " ".join(Word_List)
print('Sentence=', Sentence)
```

Output

```
'Sentence= I love data'
```

Parameters :

parameter	dtype	Argument Type	default value	description
message_d	string	compulsory		message from the first text file

message_e	string	compulsory		message from the second text file

Returns:

returns	dtype	description
final_msg	string	Sentence after applying all the above operations

- Call the function `"compare_msg()"` with `'message_4'` & `'message_5'` and store the result of it in a variable called `'secret_msg_3'`

```
# File path for message 4  and message 5
file_path_4
file_path_5

#Code starts here
message_4 = read_file(file_path_4)
message_5 = read_file(file_path_5)

print(message_4)
print(message_5)

def compare_msg(message_d,message_e):
   a_list = message_d.split()
   b_list = message_e.split()
   c_list = [i for i in a_list if i not in b_list]
   final_msg = " ".join(c_list)
   return final_msg

secret_msg_3 = compare_msg(message_4,message_5)
# File path for message 4  and message 5
file_path_4
file_path_5

#Code starts here
message_4 = read_file(file_path_4)
message_5 = read_file(file_path_5)

print(message_4)
print(message_5)

def compare_msg(message_d,message_e):
   a_list = message_d.split()
```

```
    b_list = message_e.split()
    c_list = [i for i in a_list if i not in b_list]
    final_msg = " ".join(c_list)
    return final_msg

secret_msg_3 = compare_msg(message_4,message_5)
```

Message Filter

In this task, you have to extract only those words from the message in the file that are of even length.

Instructions:

- Path for the file that you will require for this task has been stored in variables `file_path_6`
- Call the function `read_file()` for `file_path_6` and store its message sentence in variables `message_6`
- Print `message_6` to see what it contains.
- Write a function `extract_msg()` that :
    - Takes `message_f` as a parameter
    - Breaks down the sentence in `message_f` into words and stores them in `a_list`
    - Creates a lambda function called `even_word` with the condition that will return true if length of x (lambda function variable) is even
    - Implements `filter()` function with function parameter as `even_word` and sequence parameter as `a_list` and stores the result of it in a variable called `b_list`
    - Combines the words of `b_list` back to a sentence and stores it in a variable called `final_msg` and returns it

Parameters :

parameter	dtype	Argument Type	default value	description
message_f	string	compulsory		message from the text file

Returns:

returns	dtype	description
final_msg	string	Sentence after applying all the above operations

- Call the function `"extract_msg()"` with `'message_6'` and store the result of it in a variable called `'secret_msg_4'`

```
#Code starts here
```

```
message_6 = read_file(file_path_6)
print(message_6)

def extract_msg(message_f):
   a_list = message_f.split()
   even_word = lambda x : True if len(x) % 2 == 0 else False
   b_list = filter(even_word,a_list)
   final_msg = " ".join(b_list)
   return final_msg
secret_msg_4 = extract_msg(message_6)
```

Congrats lieutenant, you have successfully deciphered all the message bits that we received. In this final task, we will combine all the message bits into a single message and write it in a file.

Instructions:

- The message parts that you deciphered have been provided to you in the order that they have to be read, in a list called `message_parts`.
- Combine the contents of `message_parts` into a single sentence and store it in a variable called `secret_msg`
- Write a function `write_file()` that :
    - Takes `secret_msg` and `path`as parameters
    - Opens the file mentioned in the `path` in `a+` mode
    - Writes the content of the `secret_msg` in the above opened file
    - Closes the file

Parameters :

parameter	dtype	Argument Type	default value	description
secret_msg	string	compulsory		message from the text file
path	string	compulsory		path pointing towards the file

Returns:

The function has no return parameters

- Call the function `"write_file()"` with `'secret_msg'` and `'final_path'`

`(final_path= user_data_dir + '/secret_message.txt')`

- Print the content of the `'secret_msg'` so you can also see the message

```
#Secret message parts in the correct order
message_parts=[secret_msg_3, secret_msg_1, secret_msg_4, secret_msg_2]
```

```
final_path= user_data_dir + '/secret_message.txt'

#Code starts here

secret_msg = " ".join(message_parts)

def write_file(secret_msg,path):
   final = open(final_path,'a+')
   final.write(secret_msg)
   final.close()

write_file(secret_msg,final_path)
print(secret_msg)
```