

# CS: Objects in Python

## Polymorphism

## Lab - Polymorphism

### Polymorphism Lab 1

---

#### Lab 1

This lab will focus on using polymorphism while interacting with a contact list. There will be a main class `Contacts` that controls the “view” the user sees and responds to user input. The contact information (personal and work) will be an instance of the `Information` class.

#### The Contacts Class

This lab is built around the `Contacts` class, which has four attributes:

- \* `view` - This attribute controls what the user sees. When the value for `view` changes, the information changes. There are four different views.
- \* List view - Shows the list of all of the contacts.
- \* Information view - Shows the work and personal information for a particular contact.
- \* Add view - Add information for a new contact.
- \* Quit view - Leave a message for the user and then end the script.
- \* `contact_list` - This is a list of objects that contain the information for each contact.
- \* `choice` - This attribute represents input from the user and is used to change the view.

\* `index` - This attribute keeps track of the particular contact whose information is to be displayed.

For testing purposes, set `self.view` to `'quit'`. We will change this to a more appropriate value later on. The other attributes do not need a value when instantiating the object. Make `contact_list` an empty list, and set `choice` and `index` to `None`.

```
class Contacts:
    def __init__(self):
        self.view = 'quit'
        self.contact_list = []
        self.choice = None
        self.index = None
```

## The Display Method

The display method is designed to be a loop that runs until the user tells the script to end. The method checks the value of the `view` attribute and calls the appropriate method that displays the information for each view. Since the loop is `while True`, be sure to include a `break` statement otherwise the loop would never stop (Python would eventually stop the script with an error message).

```
def display(self):
    while True:
        if self.view == 'list':
            self.show_list()
        elif self.view == 'info':
            self.show_info()
        elif self.view == 'add':
            print()
            self.add_contact()
        elif self.view == 'quit':
            print('\nClosing the contact list...\n')
            break
```

## Starting the Other Methods

The `display` method calls three other methods. Trying to test the code would cause your script to crash as these methods have not yet been defined. We can create the methods with the `pass` statement, which allows for the code to run (`pass` is a place holder). We will come back later and replace `pass` with working code.

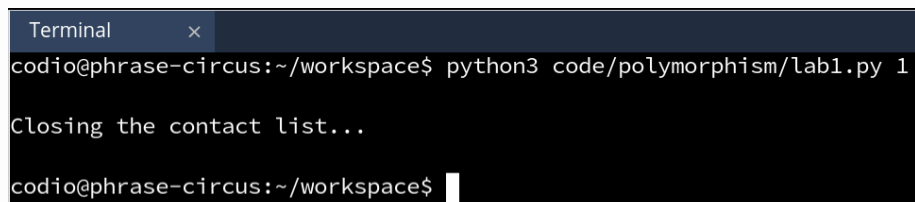
```
def show_list(self):  
    pass  
  
def show_info(self):  
    pass  
  
def add_contact(self):  
    pass
```

## Testing Your Code

Before moving on to the next part of the script, we want to check that our code is working. To do that, instantiate a `Contacts` object and call the `display` method.

```
contacts = Contacts()  
contacts.display()
```

Run your program. Because the `view` attribute is `'quit'`, the script should immediately display a message and then stop. Your output should look something like this.

A terminal window titled "Terminal" with a close button. The prompt is "codio@phrase-circus:~/workspace\$". The command "python3 code/polymorphism/lab1.py 1" has been executed. The output is "Closing the contact list...". The prompt is now "codio@phrase-circus:~/workspace\$".

```
Terminal x  
codio@phrase-circus:~/workspace$ python3 code/polymorphism/lab1.py 1  
  
Closing the contact list...  
  
codio@phrase-circus:~/workspace$
```

Lab 1 Output

▼ Code

Your code should look like this:

```
class Contacts:
    def __init__(self):
        self.view = 'quit'
        self.contact_list = []
        self.choice = None
        self.index = None

    def display(self):
        while True:
            if self.view == 'list':
                self.show_list()
            elif self.view == 'info':
                self.show_info()
            elif self.view == 'add':
                print()
                self.add_contact()
            elif self.view == 'quit':
                print('\nClosing the contact list...\n')
                break

    def show_list(self):
        pass

    def show_info(self):
        pass

    def add_contact(self):
        pass

contacts = Contacts()
contacts.display()
```

# Polymorphism Lab 2

---

## Adding a Contact

The first thing we need to do is change the default view when a `Contacts` object is instantiated. The list view should be the first view shown to the user. Change the value for `self.view` from `'quit'` to `'list'` in the constructor.

```
def __init__(self):  
    self.view = 'list'
```

Next we want to modify the `show_list` method to show the list of people in the contact list. There are two possible states for the list view: the list is empty or there are contacts in the list. When the list is empty, the user will be provided with the choice to add a contact or quit the script. Use a conditional to represent these two states. For now, set `self.view` to `'quit'` in the `else` branch.

The `print` statement is to add a blank line for legibility. If `self.contact_list` is an empty list (has a length of 0), then present the user with a choice. Store their input in `self.choice`. The `.lower()` will convert the user choice to a lowercase letter. This will make comparisons easier. Remember, Python is case sensitive; `q` and `Q` are not the same. By forcing all input to lowercase, we only need to test for the lowercase letter. The method ends by calling another method to handle the user's choice.

```
def show_list(self):  
    print()  
    if len(self.contact_list) == 0:  
        self.choice = input('(A)dd a new contact \n(Q)uit \n> ').lower()  
  
    else:  
        self.view = 'quit'  
        self.handle_choice()
```

## Handling User Choices

Every time the user makes a choice, we want to evaluate that choice and perform the appropriate action. In this case, the user can choose between adding a contact or quitting the script. Notice that `self.view` only changes to 'add' if 'a' is entered **and** we are in list view. We only want to add new contacts from the list view.

```
def handle_choice(self):
    if self.choice == 'q':
        self.view = 'quit'
    elif self.choice == 'a' and self.view == 'list':
        self.view = 'add'
```

## The Information Class

The Information class is used to store the information about a contact. These objects are elements in the `self.contact_list`. We need to define this class before we can add new contacts to the list. The constructor is going to ask the user to input the first name, last name, personal phone number, personal email, work phone number, work email, and work title. Enter the code for the Information class after the Contacts class but before `contacts` is instantiated.

```
class Information:
    def __init__(self):
        self.first_name = input('Enter their first name: ')
        self.last_name = input('Enter their last name: ')
        self.personal_phone = input('Enter their personal phone number: ')

        self.personal_email = input('Enter their personal email address: ')

        self.work_phone = input('Enter their work phone number: ')
        self.work_email = input('Enter their work email address: ')
        self.title = input('Enter their work title: ')
```

## Adding a Contact

Now that the `Information` class has been declared, we can add a new contact. To do this, we are going to modify the `add_contact` method. To add some polymorphism to the lab, we are going to overload the `+` operator to add an `Information` object to `self.contact_list`. Once the element has been added, revert back to the list view.

```
def __add__(self, new_contact):
    self.contact_list.append(new_contact)

def add_contact(self):
    self + Information()
    self.view = 'list'
```

## Testing Your Code

Before moving on to the next part of the script, we want to check that our code is adding a contact to the list. To do that, enter a when prompted, then add the following contact:

```
Rachel
Kim
555 123-4567
rachel_k@gmail.com
555 890-1234
rkim@apple.com
Senior Software Engineer
```

Then add the following code at the end of your script to check that the contact was actually added to the list. Your script should print 1 as there is one contact (“Rachel”) in the list.

```
contacts = Contacts()
contacts.display()
print(len(contacts.contact_list))
```

### ▼ Code

Your code should look like this:

```
class Contacts:
    def __init__(self):
        self.view = 'list'
        self.contact_list = []
        self.choice = None
        self.index = None

    def display(self):
        while True:
            if self.view == 'list':
                self.show_list()
            elif self.view == 'info':
                self.show_info()
            elif self.view == 'add':
                print()
                self.add_contact()
            elif self.view == 'quit':
                print('\nClosing the contact list...\n')
                break

    def show_list(self):
        print()
        if len(self.contact_list) == 0:
            self.choice = input('(A)dd a new contact \n(Q)uit \n> ').lower()

        else:
            self.view = 'quit'
            self.handle_choice()

    def show_info(self):
        pass

    def __add__(self, new_contact):
        self.contact_list.append(new_contact)

    def add_contact(self):
        self + Information()
        self.view = 'list'
```



```
def handle_choice(self):
    if self.choice == 'q':
        self.view = 'quit'
    elif self.choice == 'a' and self.view == 'list':
        self.view = 'add'

class Information:
    def __init__(self):
        self.first_name = input('Enter their first name: ')
        self.last_name = input('Enter their last name: ')
        self.personal_phone = input('Enter their personal phone number: ')

        self.personal_email = input('Enter their personal email address: ')

        self.work_phone = input('Enter their work phone number: ')
        self.work_email = input('Enter their work email address: ')
        self.title = input('Enter their work title: ')

contacts = Contacts()
contacts.display()
print(len(contacts.contact_list))
```

# Polymorphism Lab 3

---

## Displaying the List View

Now that we can add a contact to the list, we will want to show all of the contacts in the list. But before doing that, we need to remove the print statement at the end of the script. The final two lines of code should look like this:

```
contacts = Contacts()
contacts.display()
```

We are going to iterate through the list of contacts and print the first and last name. To help users select a contact, a number will appear before each name. This way, the user types the number, and the contact's information appears. In the else branch of the `show_list` method, iterate through `self.contact_list`. However, we want the index and the value of each element in the list. The index will be used for the number, and the value will be used for the first and last names. Python has the `enumerate` function which does exactly that.

Represents the value for each element in the list

```
for index, value in enumerate(my_list):
```

Represents the index for each element in the list

For each element in the list, the `enumerate` function returns the index and the value.

### Enumerate Function

Using `enumerate`, print the index plus 1 (the list should not start with 0), the `first_name`, and the `last_name` for each element. After the loop runs, ask the user if they want to select a name, add a new contact, or quit.

```

def show_list(self):
    print()
    if len(self.contact_list) == 0:
        self.choice = input('(A)dd a new contact \n(Q)uit \n> ').lower()

    else:
        for index, contact in enumerate(self.contact_list):
            print(f"{index + 1}) {contact.first_name} {contact.last_name}")

        self.choice = input('\n(#) Select a name \n(A)dd a new contact\n(Q)uit \n> ').lower()

    self.handle_choice()

```

## Handling Numeric Input

Add an `elif` branch to the `handle_choice` method that asks if the user input is numeric (remember, the `input` command stores user input as a string) and if the user is in the list view. If yes, then convert the user input to the index by converting the string to an integer and subtracting 1. Remember, we added one to the index in the `show_list` method. If the user made a mistake in entering the number, the script will crash if you try to access an index that is outside of the list. So we need to verify that the number is between 0 and the end of the list. Finally, set `self.index` to `index` and set `self.view` to `'info'`.

```

def handle_choice(self):
    if self.choice == 'q':
        self.view = 'quit'
    elif self.choice == 'a' and self.view == 'list':
        self.view = 'add'
    elif self.choice.isnumeric() and self.view == 'list':
        index = int(self.choice) - 1
        if index >= 0 and index < len(self.contact_list):
            self.index = index
            self.view = 'info'

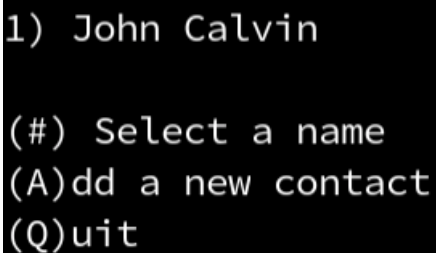
```

## Testing Your Code

Before moving on to the next part of the script, we want to check that our code is displaying all of the contacts in the list. To do that, enter two different contacts. The first one is:

```
> John  
> Calvin  
> 555 111-2222  
> john.calvin@email.net  
> 555 333-4444  
> jcalvin@work.org  
> Philosopher
```

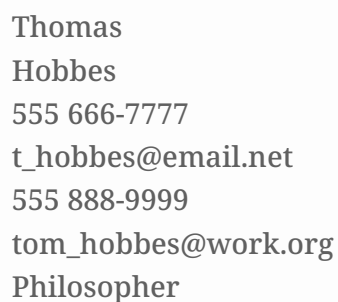
You should see a list that looks like this:



```
1) John Calvin  
  
(#) Select a name  
(A)dd a new contact  
(Q)uit
```

Contact 1

Now add a second contact to the list:



```
Thomas  
Hobbes  
555 666-7777  
t_hobbes@email.net  
555 888-9999  
tom_hobbes@work.org  
Philosopher
```

Your script should now show the following output:

```
1) John Calvin
2) Thomas Hobbes

(#) Select a name
(A)dd a new contact
(Q)uit
```

Contact 2

### ▼ Code

Your code should look like this:

```
class Contacts:
    def __init__(self):
        self.view = 'list'
        self.contact_list = []
        self.choice = None
        self.index = None

    def display(self):
        while True:
            if self.view == 'list':
                self.show_list()
            elif self.view == 'info':
                self.show_info()
            elif self.view == 'add':
                print()
                self.add_contact()
            elif self.view == 'quit':
                print('\nClosing the contact list...\n')
                break

    def show_list(self):
        print()
        if len(self.contact_list) == 0:
            self.choice = input('(A)dd a new contact \n(Q)uit \n> ').lower()

        else:
            for index, contact in enumerate(self.contact_list):
                print(f"{index + 1}) {contact.first_name} {contact.last_name}
```

```

        self.choice = input('\n(#) Select a name \n(A)dd a new contact

self.handle_choice()

def show_info(self):
    pass

def __add__(self, new_contact):
    self.contact_list.append(new_contact)

def add_contact(self):
    self + Information()
    self.view = 'list'

def handle_choice(self):
    if self.choice == 'q':
        self.view = 'quit'
    elif self.choice == 'a' and self.view == 'list':
        self.view = 'add'
    elif self.choice.isnumeric() and self.view == 'list':
        index = int(self.choice) - 1
        if index >= 0 and index < len(self.contact_list):
            self.index = index
            self.view = 'info'

class Information:
    def __init__(self):
        self.first_name = input('Enter their first name: ')
        self.last_name = input('Enter their last name: ')
        self.personal_phone = input('Enter their personal phone number:

        self.personal_email = input('Enter their personal email address:

        self.work_phone = input('Enter their work phone number: ')
        self.work_email = input('Enter their work email address: ')
        self.title = input('Enter their work title: ')

contacts = Contacts()
contacts.display()

```

# Polymorphism Lab 4

---

## Displaying Contact Info

The next step is to display the contact information for a selected contact. On the last page, we already modified `handle_choice` to deal with numeric input. So let's update the `show_info` method to display the information. Go to the appropriate element in `self.contact_list` and call the `display_info` method. Then present the user with some options to return to the list view, go to the next contact, go to the previous contact, or quit. Call the `handle_choice` method to act on the user input.

```
def show_info(self):
    self.contact_list[self.index].display_info()
    self.choice = input('\n(C)ontact List \n(P)revious contact \n(N)ext
    self.handle_choice()
```

Remember, the elements in `self.contact_list` are instances of the `Information` class. So be sure that you are defining the `display_info` in the `Information` class. This method prints out the information for the contact with a little bit of context.

```
def display_info(self):
    print(f'\n{self.first_name} {self.last_name}')
    print(f'Personal phone number: {self.personal_phone}')
    print(f'Personal email address: {self.personal_email}')
    print(f'Work title: {self.title}')
    print(f'Work phone number: {self.work_phone}')
    print(f'Work email address: {self.work_email}')
```

## Return to Contact List

We want the user to be able to list view of all contacts from the info view when the user enters c. Modify the `handle_choice` method to set `self.view` to 'list' when `self.choice` is c.

```
elif self.choice == 'c' and self.view == 'info':  
    self.view = 'list'
```

## Next and Previous Contacts

The next feature to add is the ability to change contacts (next or previous) from the info view. Moving the next contact should increase `self.index` by 1, and moving to the previous contact should decrease `self.index` by 1. However, we need to make sure that the index is not out of the range of list; this would cause the script to crash. To avoid this we are going to have a “wrapping” effect. If the index is at the end of the list, advancing to the next contact takes you back to the beginning. Similarly, going to the previous contact from the beginning of the list will take you to the end of the list.

To do this, we could write a traditional conditional. However, that is four lines of code. Instead, you can use a ternary operator (also called a conditional expression) to write the same logic in one line of code.

```
variable = value_if_true if boolean_expression else value_if_false
```

### Ternary Operator

So when the user enters n the index will increase by 1 as long as the index plus 1 is less than the length of the list. If not, the index becomes 0 (the first contact). Similarly, when the user enters p, the index will decrease by 1 as long as the index minus 1 is greater than or equal to 0. If not, the index becomes the length of the list minus 1 (the last contact). Add the following conditional branches to the `handle_choice` method.



```
elif self.choice == 'n' and self.view == 'info':
    self.index = self.index + 1 if self.index + 1 < len(self.contact_list) else 0

elif self.choice == 'p' and self.view == 'info':
    self.index = self.index - 1 if self.index - 1 >= 0 else len(self.contact_list) - 1
```

## Testing Your Code

This script should be complete now. To test it, add at two contacts. Pull up their information and use n and p to cycle through the contact information. Enter c to return to the list view of all the contacts.

### ▼ Code

Your code should look like this:

```
class Contacts:
    def __init__(self):
        self.view = 'list'
        self.contact_list = []
        self.choice = None
        self.index = None

    def display(self):
        while True:
            if self.view == 'list':
                self.show_list()
            elif self.view == 'info':
                self.show_info()
            elif self.view == 'add':
                print()
                self.add_contact()
            elif self.view == 'quit':
                print('\nClosing the contact list...\n')
                break

    def show_list(self):
        print()
        if len(self.contact_list) == 0:
```

```

        self.choice = input('(A)dd a new contact \n(Q)uit \n> ').lower

    else:
        for index, contact in enumerate(self.contact_list):
            print(f"{index + 1}) {contact.first_name} {contact.last_name}")

        self.choice = input('\n(#) Select a name \n(A)dd a new contact \n(Q)uit \n> ').lower

    self.handle_choice()

def show_info(self):
    self.contact_list[self.index].display_info()
    self.choice = input('\n(C)ontact List \n(P)revious contact \n(N)ext \n> ').lower

    self.handle_choice()

def __add__(self, new_contact):
    self.contact_list.append(new_contact)

def add_contact(self):
    self + Information()
    self.view = 'list'

def handle_choice(self):
    if self.choice == 'q':
        self.view = 'quit'
    elif self.choice == 'a' and self.view == 'list':
        self.view = 'add'
    elif self.choice.isnumeric() and self.view == 'list':
        index = int(self.choice) - 1
        if index >= 0 and index < len(self.contact_list):
            self.index = index
            self.view = 'info'
    elif self.choice == 'c' and self.view == 'info':
        self.view = 'list'
    elif self.choice == 'n' and self.view == 'info':
        self.index = self.index + 1 if self.index + 1 < len(self.contact_list) else len(self.contact_list) - 1
    elif self.choice == 'p' and self.view == 'info':
        self.index = self.index - 1 if self.index - 1 >= 0 else len(self.contact_list) - 1

```

```
class Information:
    def __init__(self):
        self.first_name = input('Enter their first name: ')
        self.last_name = input('Enter their last name: ')
        self.personal_phone = input('Enter their personal phone number: ')

        self.personal_email = input('Enter their personal email address: ')

        self.work_phone = input('Enter their work phone number: ')
        self.work_email = input('Enter their work email address: ')
        self.title = input('Enter their work title: ')

    def display_info(self):
        print(f'\n{self.first_name} {self.last_name}')
        print(f'Personal phone number: {self.personal_phone}')
        print(f'Personal email address: {self.personal_email}')
        print(f'Work title: {self.title}')
        print(f'Work phone number: {self.work_phone}')
        print(f'Work email address: {self.work_email}')

contacts = Contacts()
contacts.display()
```

# Polymorphism Lab Challenge

---

## Problem

In the IDE to the left, the class `Chef` is already defined. Modify the `compare` method so that it uses operator overloading to compare two chefs and determine who has more Michelin stars. You can use either the `>` or `<` operators for your overloading.

## Expected Output

Instantiate the following `Chef` objects:

```
marco = Chef('Marco Pierre White', 'French, British', 3)
rene = Chef('Rene Redzepi', 'Nordic', 2)
```

Your script should return the proper output no matter the order of comparison.

Method Call	Return Value
<code>marco.compare(rene)</code>	'Marco Pierre White has more Michelin stars than Rene Redzepi'
<code>rene.compare(marco)</code>	'Marco Pierre White has more Michelin stars than Rene Redzepi'