Amr Bahaa Eldin Mohamed Ahmed Ali

4200491

Python Special_Dunder Method

Implementing dive techniques for classes is a good form of polymorphism. If you have created a class in Python and use the init function, you are already using dunder methods.

Before we continue it will be important to have the following:

- A basic understanding of Object Oriented Programming using Python.
- Experience working with classes in Python.
- Before we continue it will be important to have the following:
- A basic understanding of Object Oriented Programming using Python.
- Experience working with classes in Python.

Dunder methods for our class

- Init

  This is a method you must have already used if you have worked with classes. The init method is used to create an instance of the class.

  example

```
def __init__(self,names):
    if names:
        self.names = names.copy()
        for name in names:
            self.versions[name] = 1
```

```
    else:
        raise Exception("Please Enter the names")
```

- **Str**

  The str method is useful when we want to use instances of our class in a print statement.

  Example

```
def __str__(self):
    s ="The current softwares and their versions are listed
below: \n"
    for key,value in self.versions.items():
        s+= f"{key} : v{value} \n"
    return s
```

- **setitem**

  When assigning values in a dictionary, the setitem method is invoked. The method above is going to update the version number of the software. If the software is not found, it will raise an error.

  example

```
def __setitem__(self,name,version):
    if name in self.versions:
        self.versions[name] = version
    else:
        raise Exception("Software Name doesn't exist")
```

- **getitem**

  The getitem method is like the setitem method, the major difference being that the getitem method is called when we use the [] operator of a dictionary.

Example

```
def __getitem__(self,name):
    if name in self.versions:
        return self.versions[name]
    else:
        raise Exception("Software Name doesn't exist")
```

- **delitem**

The delitem is like the setitem and getitem method. To avoid repetition, we will move on to the implementation and use case.

```
def __delitem__(self,name):
    if name in self.versions:
        del self.versions[name]
        self.names.remove(name)
    else:
        raise Exception("Software Name doesn't exist")
```

- **len**

In a dictionary, the len method returns the number of elements in a list or the number of key-value pairs in a dictionary.

```
def __len__(self):
    return len(self.names)
```

- **contains**

The contains method is used when using the in operator. The return value has to be a boolean.

```
def __contains__(self,name):
    if name in self.versions:
        return True
    else:
        return False
```

- **Add**

The add method is called when using the + operator. We can define a custom add method for our class.

```
def __add__(self,p2):
    x = self.x + p2.x
    y = self.y + p2.y
    return point(x,y)
```