Python 101

Lec07

Classes Continued

thoum

May 21, 2019

Inheritance

One usage of classes is Inheritance.

Inheritance

The child inherits every thing about its parent, and $+\alpha$.

Inheritance

```
class MyList(list):
    """a list that keeps track of max and min"""
    def __init__(self, iterable):
        super().__init__(iterable)
        self._max = max(iterable)
        self._min = min(iterable)

    # always return 100, just for demonstrational purposes
    def __len__(self):
        return 100

    def append(self, item):
        if item < self._min:
            self._min = item
        if item > self._max:
            self._max = item
        super().append(item)


a = MyList((1,2,3,5,10))
a.insert(3, 4)
a.append(100)
print(a[3:4])
print(a)
print(a._max)
# overriden by the child
```

Explained

MyList(*list*) means that this class will inherit from *list*.

```
class MyList(list):
```

Explained

The *super*() returns the parent class. We use super() to access the parent classes data methods etc. Here, we initiate the parent first so that parameters are automatically filled in.

```
class MyList(list):
    """a list that keeps track of max and min"""
    def __init__(self, iterable):
        super().__init__(iterable)
```

Explained

$min(lst), max(lst)$ takes time proprotional to N.
Here, we keep track of min and max so that it can be known regardless of size.
(Of course, there is no free lunch, there is extra cost of comparing at append.)

```
class MyList(list):
    """a list that keeps track of max and min"""
    def __init__(self, iterable):
        super().__init__(iterable)
        self._max = max(iterable)
        self._min = min(iterable)
```

Explained

Here, we override the parent's __len__, which determines the value returned when we do *len*(*lst*).

```
    # always return 100, just for demonstrational purposes
    def __len__(self):
        return 100
```

Explained

We override the append() of list, so that we keep track of _min and _max.
After updating _min and _max, we insert to the list via super() call.

```
    def append(self, item):
        if item < self._min:
            self._min = item
        if item > self._max:
            self._max = item
        super().append(item)
```

Question

Are we done implementing MyList so that it correctlys keep track of min and max?

Question

NO

The Catch

Everything that can be done to a *list* can be done to a *MyList*. $pop()$, *del*, $insert()$, $mylst[3] = 4$, $mylst[3:4]$... you name it.

This means that to correctly keep track of min & max, we have to override *every single* method of a list that is capable of changing its contents.

Can you remember all of them?

Composition over Inheritance[1]

So, it is often wise to compose your class with a list, rather than inheriting it.

[1] Look this up in Google

MyList Composition Ver.

We can control the methods we provide for interaction with the internal list.

```
class MyList():
    def __init__(self, iterable=[]):
        self.lst = list(iterable)
        self._max = max(self.lst)
        self._min = min(self.lst)

    def __len__(self):
        return len(self.lst)

    def __repr__(self):
        return str(self.lst)

    def __getitem__(self, index):
        return self.lst[index]

    # provide an interface for our list
    def append(self, item):
        if item < self._min:
            self._min = item
        if item > self._max:
            self._max = item
        self.lst.append(item)
```

Inheritance?

We might use inheritance when the child has to provide every method its parents provide $+ \alpha$.
(*i.e.* The Gun class that inherits the Weapon class in an RPG game?).
But Design Patterns are a complicated subject by itself, so we won't deal it in detail.

Questions?

Questions on homeworks?

For Next Week

Twitter Crawler

Demonstration

(I will not post the code due to security reasons.)

For Next Week

If we have time, we do the prerequisites together. Else, it's homework.