Object Oriented Programming

Roudranil Das

PDSP Aug-Nov 2022
Roll No: MDS202227
roudranil@cmi.ac.in

September 19, 2022

Contents

1 September 7, 2022

1.1 Classes and objects

- Abstract data type
  - Stores some info
  - Designated functions to manipulate the info
  - for instance, for list: `append()`, `insert()`, `delete()`
- separate the private implementation from the public specification
- Class
  - template for a data type
  - how data is stored
  - how public functions manipulate data
- Objects
  - concrete instance of template

Initialising a class:

```python
class Something:
    def __init__(self, param):
        self.param = param
    def do_something(self):
        print(f"Hello {self.param}")
```

Using this class:

```python
obj = Something("world")
obj.do_something()
# Hello world
```

2 September 13, 2022

2.1 Default and keyword arguments

Arguments in a function are listed in some order. The default arguments are at the end, and are preceeded by those which are not. So while calling the function, then we have to provide the default arguments at the end. If we put it at the beginning, and then we pass another argument after that without actually calling the parameter name, then we get an error. If we pass arguments without calling the parameter name, then the values are assigned in the order of definition in the function definition. Default arguments are also sometimes called optional arguments.

Note that in PYTHON, the functional parameters dont have data types. Also for this reason, function overloading is also not supported in PYTHON.

2.2 Special Functions

- `__init__()` $\rightarrow$ constructor
- `__str__()` $\rightarrow$ converts object to string.
  - `o.__str__() == __str__(o)`
  - Implicitly invoked by `print()`
- `__add__()`
  - type dependent.
  - Implicitly invoked by `+`
- There are also many more, for the relational operators and others.
  - `__mult__()` $\rightarrow \times$
  - `__lt__()` $\rightarrow <$
  - `__gt__()` $\rightarrow >$

    .
    .
    .

2.3 Designing a flexible list

- Typically a sequence of nodes
- Each node contains a value and points to the next node in the sequence
- Easy to modify: insertion and deletion is easy via local plumbing, flexible size
- need to follow links to access `A[i]`: takes time $O(i)$

```python
class Node:
    def __init__(self, v = None):
        self.value = v
        self.next = None
        return

    def isempty(self):
        if self.value == None:
            return True
        else:
            return False
```

What is an empty list? `l = []` $\neq$ `l = None`. The only legal empty list is the one with both value and address `None`.

- list is a seq of nodes
  - `self.value` is the stored value
  - `self.next` points to the next node
- Empty list? `self.value` is `None`
- Creating lists
  - `L1 = Node()` $\rightarrow$ Empty list
  - `L2 = Node(5)` $\rightarrow$ Singleton list

Appending a value to a list

- add `v` to the end of the list
- if `l` is empty update `l.value()` from `None` to `v`
- if at last value, then `l.next()` is `None`. In that case, point `next` to `v`
- Otherwise recursively append to the rest of the list.
- There is an iterative implementation, where we walk down the list, node by node and then append to the last node.

3 September 15, 2022

3.1 Designing a flexible list (cotd)

Inserting a value at the list

- want to insert `v` at the head

- create a new node with `v`
- cannot change where the head points. This is because outside the function, the old self will remain unchanged.
- exchange the values v_0 and `v`
- make new node point to `head.next`
- Make `head.next` point to the new node
- basically first swap the values, and then the addresses

Appending a value to a list

- create a new node with `v`
- exchange the values v_0 and `v`
- make `self.next → newnode`
- make `newnode.next → (self.next)` (basically point it to the node where `self.next` is pointing)
- by `self`, i mean `head`.

Delete a value v

- remove the first occurance of v
- scan list for `v` - look ahead at next node
- if next node value is `v`, bypass node
- cannot bypass the first node in the list
  - instead copy the second node value to head
  - bypass head
- recursive implementation