Classes and objects

Madhavan Mukund

https://www.cmi.ac.in/~madhavan

Programming and Data Structures with Python

Lecture 11, 19 Sep 2023

Classes and objects

- Abstract datatype
  - Stores some information
  - Designated functions to manipulate the information
  - For instance, list: `append()`, `insert()`, `delete()`, ...

Classes and objects

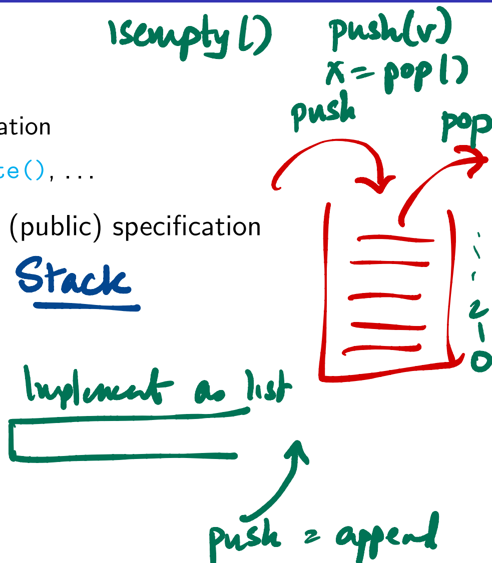
- Abstract datatype
  - Stores some information
  - Designated functions to manipulate the information
  - For instance, list: `append()`, `insert()`, `delete()`, ...
- Separate the (private) implementation from the (public) specification

Classes and objects

- Abstract datatype
  - Stores some information
  - Designated functions to manipulate the information
  - For instance, 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

Classes and objects

- Abstract datatype
  - Stores some information
  - Designated functions to manipulate the information
  - For instance, 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
- Object
  - Concrete instance of template



Example: 2D points

- A point has coordinates (x, y)
  - `__init__()` initializes internal values `x`, `y`
  - First parameter is always `self`

```
class Point:
  def __init__(self,a,b):
    self.x = a
    self.y = b
```

Example: 2D points

- A point has coordinates (x, y)
  - `__init__()` initializes internal values `x`, `y`
  - First parameter is always `self`
- Translation: shift a point by $(\Delta x, \Delta y)$
  - $(x, y) \mapsto (x + \Delta x, y + \Delta y)$

```
class Point:
  def __init__(self,a,b):
    self.x = a
    self.y = b

  def translate(self,deltax,deltay):
    self.x += deltax
    self.y += deltay
```

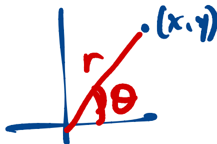
Example: 2D points

- A point has coordinates (x, y)
  - `__init__()` initializes internal values `x`, `y`
  - First parameter is always `self`
- Translation: shift a point by $(\Delta x, \Delta y)$
  - $(x, y) \mapsto (x + \Delta x, y + \Delta y)$
- Distance from the origin
  - $d = \sqrt{x^2 + y^2}$

```
class Point:
  def __init__(self,a,b):
    self.x = a
    self.y = b

  def translate(self,deltax,deltay):
    self.x += deltax
    self.y += deltay

  def odistance(self):
    import math
    d = math.sqrt(self.x*self.x +
                  self.y*self.y)
    return(d)
```

Polar coordinates

- (r, θ) instead of (x, y)
  - $r = \sqrt{x^2 + y^2}$
  - $\theta = \tan^{-1}(y/x)$




```
import math
class Point:
  def __init__(self,a,b):
    self.r = math.sqrt(a*a + b*b)
    if a == 0:
      if b >= 0:
        self.theta = math.pi/2
      else:
        self.theta = 3*math.pi/2
    else:
      self.theta = math.atan(b/a)
```

p = Point(3,5)

Polar coordinates

- (r, θ) instead of (x, y)
  - $r = \sqrt{x^2 + y^2}$
  - $\theta = \tan^{-1}(y/x)$
- Distance from origin is just r

```
import math
class Point:
  def __init__(self,a,b):
    self.r = math.sqrt(a*a + b*b)
    if a == 0:
      if b >= 0:
        self.theta = math.pi/2
      else:
        self.theta = 3*math.pi/2
    else:
      self.theta = math.atan(b/a)

  def odistance(self):
    return(self.r)
```

Polar coordinates

- (r, θ) instead of (x, y)
  - $r = \sqrt{x^2 + y^2}$
  - $\theta = \tan^{-1}(y/x)$
- Distance from origin is just r
- Translation
  - Convert (r, θ) to (x, y)
  - $x = r\cos\theta$, $y = r\sin\theta$
  - Recompute r, θ from $(x + \Delta x, y + \Delta y)$

```
def translate(self,deltax,deltay):
  x = self.r*math.cos(self.theta)
  y = self.r*math.sin(self.theta)
  x += deltax
  y += deltay
  self.r = math.sqrt(x*x + y*y)
  if x == 0:
    self.theta = math.pi/2
  else:
    self.theta = math.atan(y/x)
```

p.translate(2,4)

Polar coordinates

- (r, θ) instead of (x, y)
  - $r = \sqrt{x^2 + y^2}$
  - $\theta = \tan^{-1}(y/x)$
- Distance from origin is just r
- Translation
  - Convert (r, θ) to (x, y)
  - $x = r\cos\theta$, $y = r\sin\theta$
  - Recompute r, θ from $(x + \Delta x, y + \Delta y)$
- Interface has not changed
  - User need not be aware whether representation is (x, y) or (r, θ)

```
def translate(self,deltax,deltay):
  x = self.r*math.cos(self.theta)
  y = self.r*math.sin(self.theta)
  x += deltax
  y += deltay
  self.r = math.sqrt(x*x + y*y)
  if x == 0:
    self.theta = math.pi/2
  else:
    self.theta = math.atan(y/x)
```

Special functions

- `__init__()` — constructor

Special functions

- `__init__()` — constructor
- `__str__()` — convert object to string
  - `str(o) == o.__str()__`
  - Implicitly invoked by `print()`

```
class Point:

  ...

  def __str__(self):
    return(
      '('+str(self.x)+','
        +str(self.y)+')'
    )
```

Special functions

- `__init__()` — constructor
- `__str__()` — convert object to string
  - `str(o) == o.__str()__`
  - Implicitly invoked by `print()`
- `__add__()`
  - Implicitly invoked by `+`

```
class Point:

  ...

  def __str__(self):
    return(
      '('+str(self.x)+','
         +str(self.y)+')'
    )

  def __add__(self,p):
    return(Point(self.x + p.x,
                 self.y + p.y))
```

p1 + p2

p1.__add__(p2)

p = Point(..)
return(p)

Special functions

- `__init__()` — constructor
- `__str__()` — convert object to string
  - `str(o) == o.__str()__`
  - Implicitly invoked by `print()`
- `__add__()`
  - Implicitly invoked by `+`
- Similarly
  - `__mult__()` invoked by `*`
  - `__lt__()` invoked by `<`
  - `__ge__()` invoked by `>=`
  - ...

```
class Point:

  ...

  def __str__(self):
    return(
      '('+str(self.x)+','
          +str(self.y)+')'
    )

  def __add__(self,p):
    return(Point(self.x + p.x,
                 self.y + p.y))
```

Designing a flexible list

Madhavan Mukund

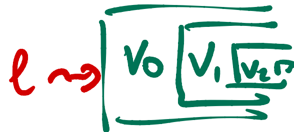
https://www.cmi.ac.in/~madhavan

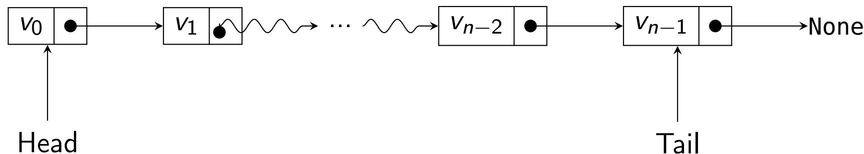
Programming and Data Structures with Python

Lecture 11, 19 Sep 2023

Lists

- Typically a sequence of nodes
- Each node contains a value and points to the next node in the sequence
  - "Linked" list
- Easy to modify
  - Inserting and deletion is easy via local "plumbing"
  - Flexible size
- Need to follow links to access `A[i]`
  - Takes time $O(i)$





- Python class Node

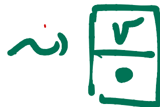
```
class Node:
    def __init__(self, v = None):
        self.value = v
        self.next = None
        return

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

Implicit initialization

int(s) ~> convert to base 10

int(s,b) ~> convert to base b

def __init__(self, a=0, b=0):



Implementing lists in Python

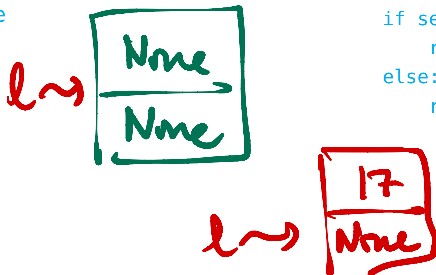
- Python class `Node`
- A list is a sequence of nodes
  - `self.value` is the stored value
  - `self.next` points to next node

```
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)
```

Implementing lists in Python

- Python class `Node`
- A list is a sequence of nodes
  - `self.value` is the stored value
  - `self.next` points to next node
- Empty list?
  - `self.value` is `None`

```
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)
```



Implementing lists in Python

- Python class `Node`
- A list is a sequence of nodes
  - `self.value` is the stored value
  - `self.next` points to next node
- Empty list?
  - `self.value` is `None`
- Creating lists
  - `l1 = Node()` — empty list
  - `l2 = Node(5)` — singleton list

l3 = Node([3,6,9])

```
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)
```

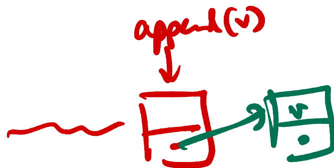
Implementing lists in Python

- Python class Node
- A list is a sequence of nodes
  - self.value is the stored value
  - self.next points to next node
- Empty list?
  - self.value is None
- Creating lists
  - l1 = Node() — empty list
  - l2 = Node(5) — singleton list
  - l1.isempty() == True
  - l2.isempty() == False

```
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)
```

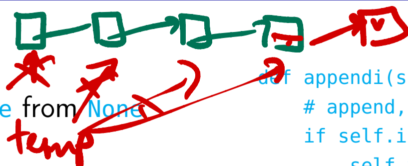
Appending to a list

- Add v to the end of list l
- If l is empty, update l.value from None to v
- If at last value, l.next is None
  - Point next at new node with value v
- Otherwise, recursively append to rest of list




```
def append(self,v):
    # append, recursive
    if self.isempty():
        self.value = v
    elif self.next == None:
        self.next = Node(v)
    else:
        self.next.append(v)
    return
```

Appending to a list

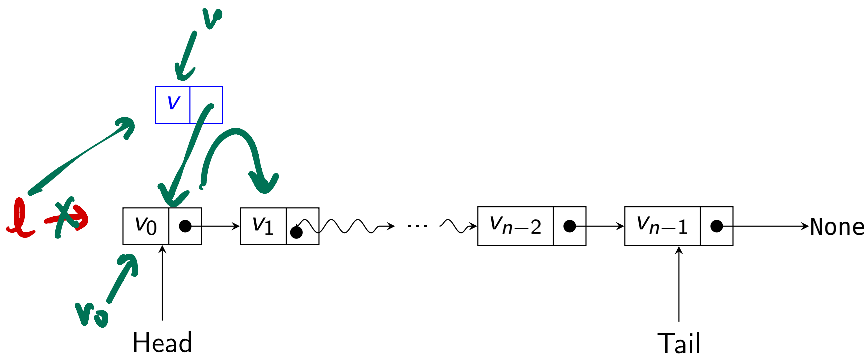


- Add v to the end of list l
- If l is empty, update l.value from None to v
- If at last value, l.next is None
  - Point next at new node with value v
- Otherwise, recursively append to rest of list
- Iterative implementation
  - If empty, replace l.value by v
  - Loop through l.next to end of list
  - Add v at the end of the list

```
def appendi(self,v):
    # append, iterative
    if self.isempty():
        self.value = v
        return

    temp = self
    while temp.next != None:
        temp = temp.next

    temp.next = Node(v)
    return
```

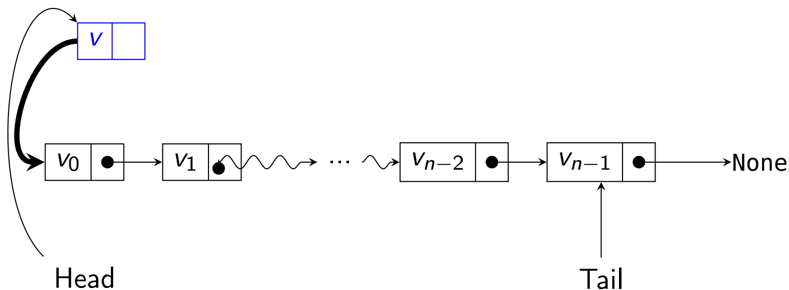
Insert at the start of the list

- Want to insert v at head
- Create a new node with v



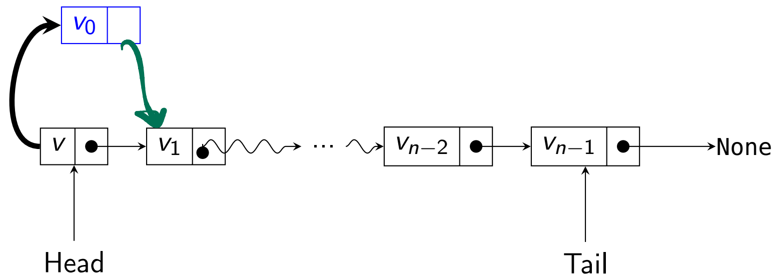
Insert at the start of the list

- Want to insert v at head
- Create a new node with v
- Cannot change where the head points!



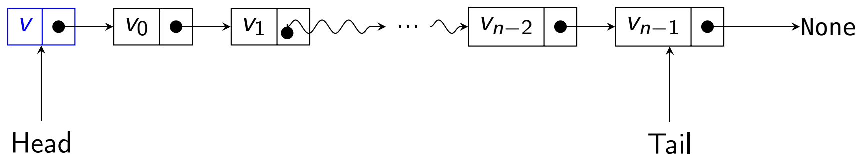
Insert at the start of the list

- Want to insert v at head
- Create a new node with v
- Cannot change where the head points!
- Exchange the values v_0, v



Insert at the start of the list

- Want to insert v at head
- Create a new node with v
- Cannot change where the head points!
- Exchange the values v_0, v
- Make new node point to `head.next`
- Make `head.next` point to new node



Inserting in a list

- Create a new node with v
- Exchange the values v_0, v
- Make new node point to `head.next`
- Make `head.next` point to new node

```
def insert(self,v):
    if self.isempty():
        self.value = v
        return

    newnode = Node(v)

    # Exchange values in self and newnode
    (self.value, newnode.value) =
        (newnode.value, self.value)

    # Switch links
    (self.next, newnode.next) =
        (newnode, self.next)

    return
```