Inductive definitions

$0! = 1$

$n! = n * (n-1)!$

Base case

$f(0) = v$

$f(n) = g(n, f(n-1))$

↑ simple fn.

$4!$? $\rightarrow 4 * 3!$

$\rightarrow 4 * (3 * 2!)$

$\rightarrow 4 * (3 * (2 * (1!)))$

$\rightarrow 4 * (3 * (2 * (1 * 0!)))$ ← known = 1

$f(0) = 0$

$f(1) = 1$

$f(n) = f(n-1) + f(n-2)$

0	1	2	3	4
0	1	1	2	3

$f(4) = f(3) + f(2)$

$= (f(2) + f(1)) + (f(1) + f(0))$

$f(2) = f(1) + f(0)$

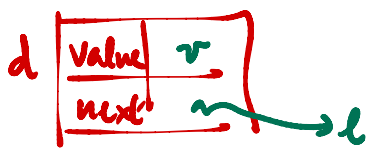
$1 \quad 0$

$= 1 + 1 + 1 + 0 = 3$

Induction on structures

List

Dictionary implementation



l = l[:1] + l[1:]

l[0] rest

[l[0]] + l[1:]

Base Case : []

Length of a list

$$\text{length}([]) = 0$$

$$\text{length}([l[0]] + l[1:]) = 1 + \text{length}(l[1:])$$

$$\begin{aligned}
\text{length}([1,2,3]) &\rightsquigarrow 1 + \text{length}([2,3]) \\
&\rightsquigarrow 1 + 1 + \text{length}([3]) \\
&\rightsquigarrow 1 + 1 + 1 + \text{length}([]) = 3
\end{aligned}$$

Sum

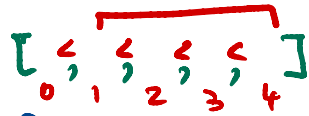
$$sum([]) = 0$$

$$sum(l) = l[0] + sum(l[1:])$$

Ascending order

$l[0] < l[1] < \ldots$

```
if len(l) <= 1:
    return (True)
else:
    return (l[0]<l[1] and
            ascending(l[1:]))
```




```
for i in range(len(l)-1):
    if l[i] >= l[i+1]:
        return (False)
return(True)
```

empty list : len is 0

Singleton

Zigzag

Base Case(s)?

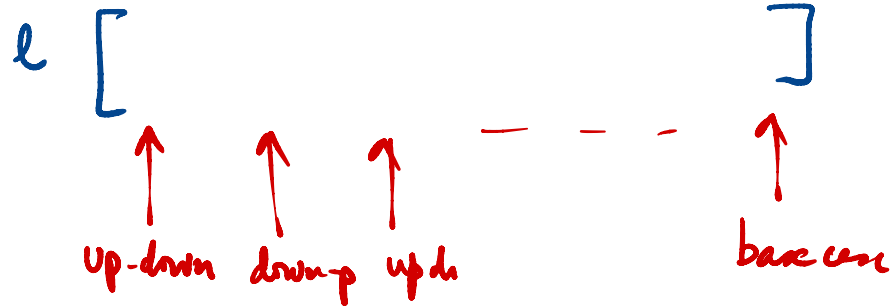
len = 0 ✓ True

len = 1 ✓ True

len = 2 ? Check l[0] == l[1] !

$l[0] < l[1] \rightsquigarrow$ down-up $(l[1:])$

$l[0] > l[1] \rightsquigarrow$ up-down $(l[1:])$


l
up-down
down-up
up-d
base case

Classes and objects

Madhavan Mukund

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

Programming and Data Structures with Python

Lecture 10, 14 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

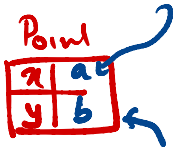
Classes and objects

- 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
```

x = x + deltax

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
  - $d = \sqrt{x^2 + y^2}$

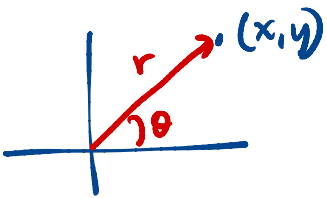
```
class Point:
  def __init__(self,a,b):
    self.x = a
    self.y = b

  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)
```

(x,y) coord

Internal change due to implementation change

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)
```

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)
```

p1 = Point(5,4)
p2 = Point(3.5, 6.7)