Discussion 04

Tree Recursion, Python Lists

Aditya Balasubramanian
aditbala [at] berkeley [dot] edu

Slides available at `teaching.aditbala.com`

Announcements 

- Homework 3 deadline extended to Friday 2/17.
- No office hours on Friday, so finish it by Thursday if you want help.
- Project 2 (C.A.T.S.) is due Friday 2/24.
  - Checkpoint (Phase 1) due Tuesday 2/21.
  - Early submission bonus point for finishing by Thursday 2/23.

Tree Recursion

Tree Recursion

- What is Tree Recursion?
  - Recursion, but with more recursive calls
  - Can break down the problem in more than one way
  - With all of the options drawn out, looks like a tree of recursive calls
- When and Why?
  - Useful when the original problem can be broken down in multiple ways
  - Accumulate all sub-problems with multiple recursive calls

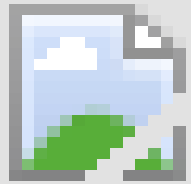
Recursive Fibonacci

```
def fib(n):
  if n == 0:
    return 0
  elif n == 1:
    return 1
  else:
    return fib(n - 1) + fib(n - 2)
```

- Need to look at `fib(n - 1)` and `fib(n-1)`
- All steps of recursion present
  - Base Case
  - Recursive Calls
  - Applying to solve problem

`Q1` , `Q2`

Lists



Lists

- An indexed collection of any data type
- Examples of valid lists:
  - `list_of_ints = [1, 2, 3, 4]`
  - `list_of_bools = [True, True, False, False]`
  - `nested_lists = [1, [2, 3], [4, [5]]]`

Creation and Usage

- In order to access the values in our list, we have to use the index
- Python lists are zero indexed, so the first element is at index 0
- `n` element is at `n-1` index
- Can also access elements in reverse order through negative index
  - Last element is accessed through index `-1` or `len(list) - 1`

```
>>> a = [1, 2, [3, 4]]
>>> a[0]
1

>>> a[2]
[3, 4]

>>> a[2][0]
3
```

Q3: WWPD (Lists)

List Slicing

- How do you access a subset of the list?
- List slicing: creating a copy of part of the list
  - Syntax: `list[<start index>: <non inclusive end index>: <step size>]`
  - step size by default is 1
  - negative step size means list is reversed

List Slicing Examples

```
>>> a = [7, 89, True, ['cat']]

>>> a[1:3]
[89, True]

>>> a[:3:2]
[7, True]

>>> a[::-1]
[['cat'], True, 89, 7]

>>> a[:2:-1] # go backwards until STOP index
[['cat']]
```

List Comprehension

- How do you create a list that fits some criteria?
  e.g. How would you create a list with numbers 1...4, but squared
  `[1, 4, 9, 16]`
- List Comprehension: creating a list based on expressions filtering other lists
- Syntax: `[<expression> for <value> in <sequence> [if <filter>]]`
- `if` condition is optional

List Comprehension Examples

```
>>> a = [x**2 for x in range(1, 5)]

>>> a
[1, 4, 9, 16]

>>> [x/2 for x in [x for x in a if x % 2 == 0]]
[1, 8]
```

Q3, Q4, Q5

Dictionaries 

- Maps `keys` to `values`
- `keys` must be immutable, `values` can be mutable
- Doesn't really have an order
- Access elements using `keys` rather than indices
- Defined with curly braces (`{}`)
  - `{key: value}`

Demo:

```
pokemon = {'pikachu': 25, 'dragonair': 148, 25: 'hello'}
pokemon['pikachu'] = pokemon['pikachu'] + 1 # 25
pokemon['mew'] = 2500
pokemon # {'pikachu': 25, 'dragonair': 148, 25: 'hello', 'mew': 2500}
```

Q6: WWPD (Dictionaries)

Thank you

Anon Feedback -> https://tinyurl.com/adit-anon