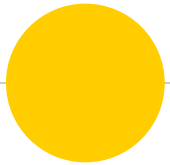
Problem solving with Python

aka MSAN 689

by Laurynas Riliskis



Last Christmas

recap

Data science in real world

- Direct business value
- Learn from less data
- Protect data and insights
- Traceable, explainable, & fair

Designing solution

Solving a data science problems goes beyond the finding the answers.

- clarity
- well-documented code
- readability of the implementation
- visibility of the of the solution design

Designing solution

- how does the data flow through the app?
- what blocks of code?
- what they will talk to?
- what is the output?

To OO or not to OO

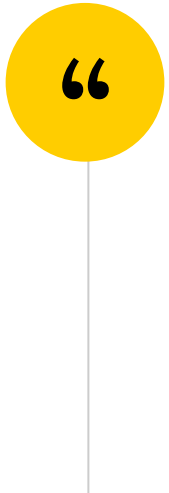
Object Oriented Programming

- Class -> blueprint
- Object -> instance
- For the specific entity, object encapsulates
  - Data
  - Attributes
  - Methods
- Used in libraries, great for abstraction and hiding.

Lecture 2

Data structures and Big O

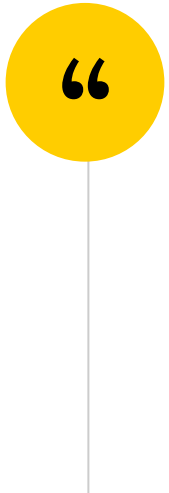
types



Data thingies

- Types
  - Basic building block
  - Composite types
- Build in in most of programming languages
- In Python, mutable vs immutable

Jupyter: types

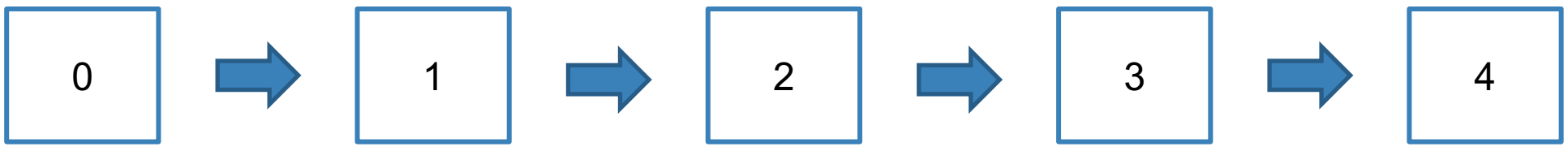


Data structures: Why, what?

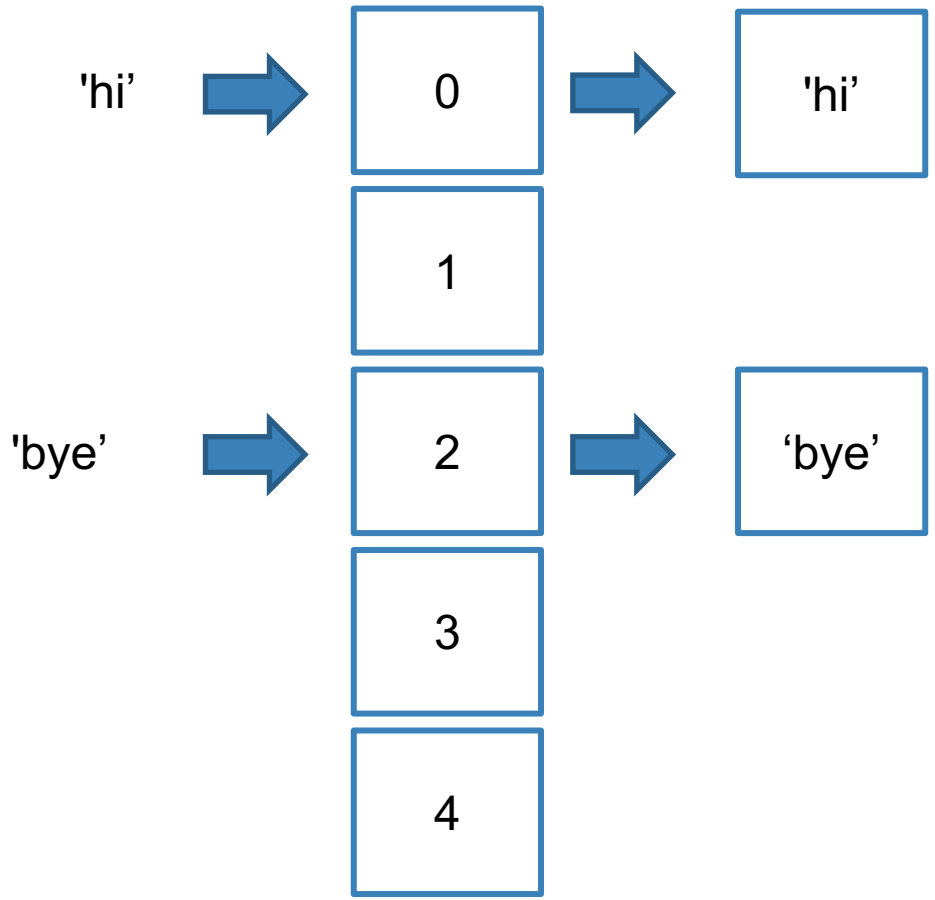
Data structures

- Is a way to organize your data
  - Insert
  - Search
  - Delete
  - Query
  - Update
- Does data structure solve problems?

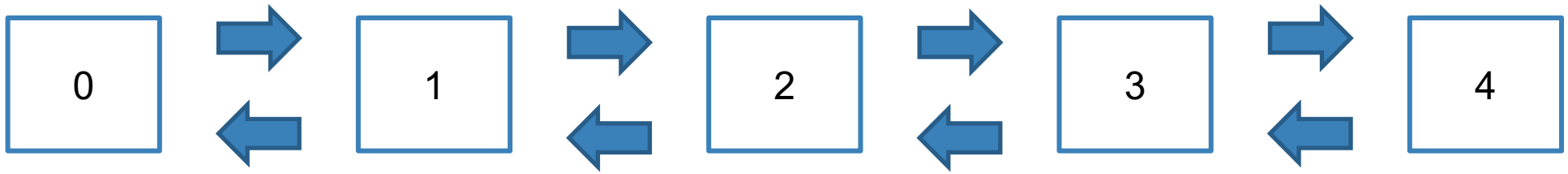
List



Hashtable



Double Linked List



Linear data structures

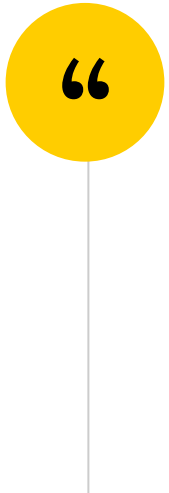
- Array – like list but constrained
- Contains:
  - Stacks – Last-In-First-Out (LIFO)
  - Queues – First-In-First-Out (FIFO)

Data structures

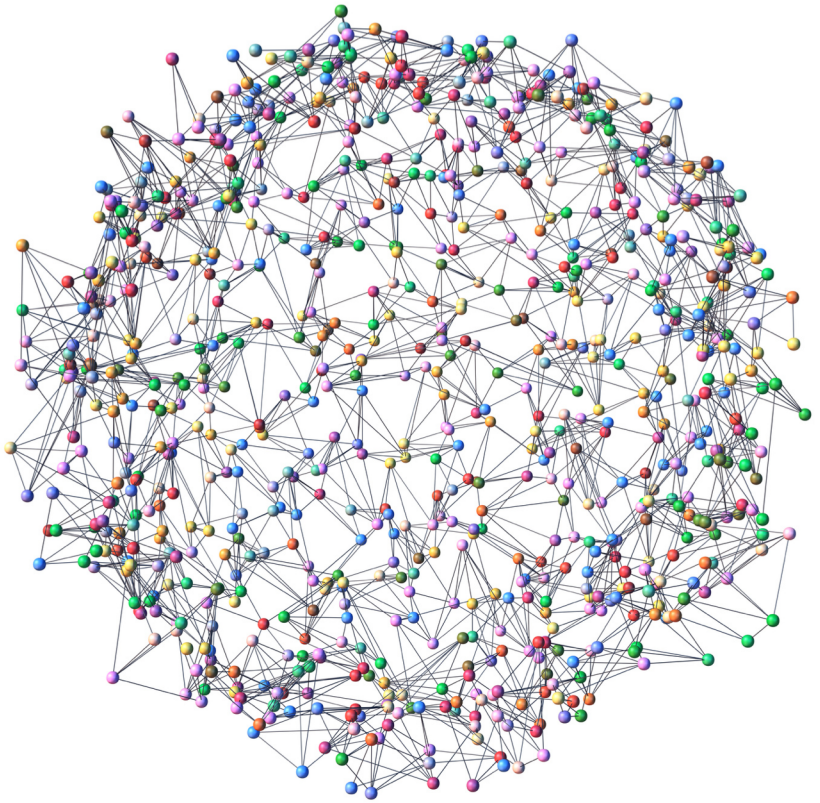
- Using the right data structure makes whole difference in time and space it takes to solve the problem

- You need to have familiarity with the libraries to understand how to apply them for your problems

Jupyter: lists



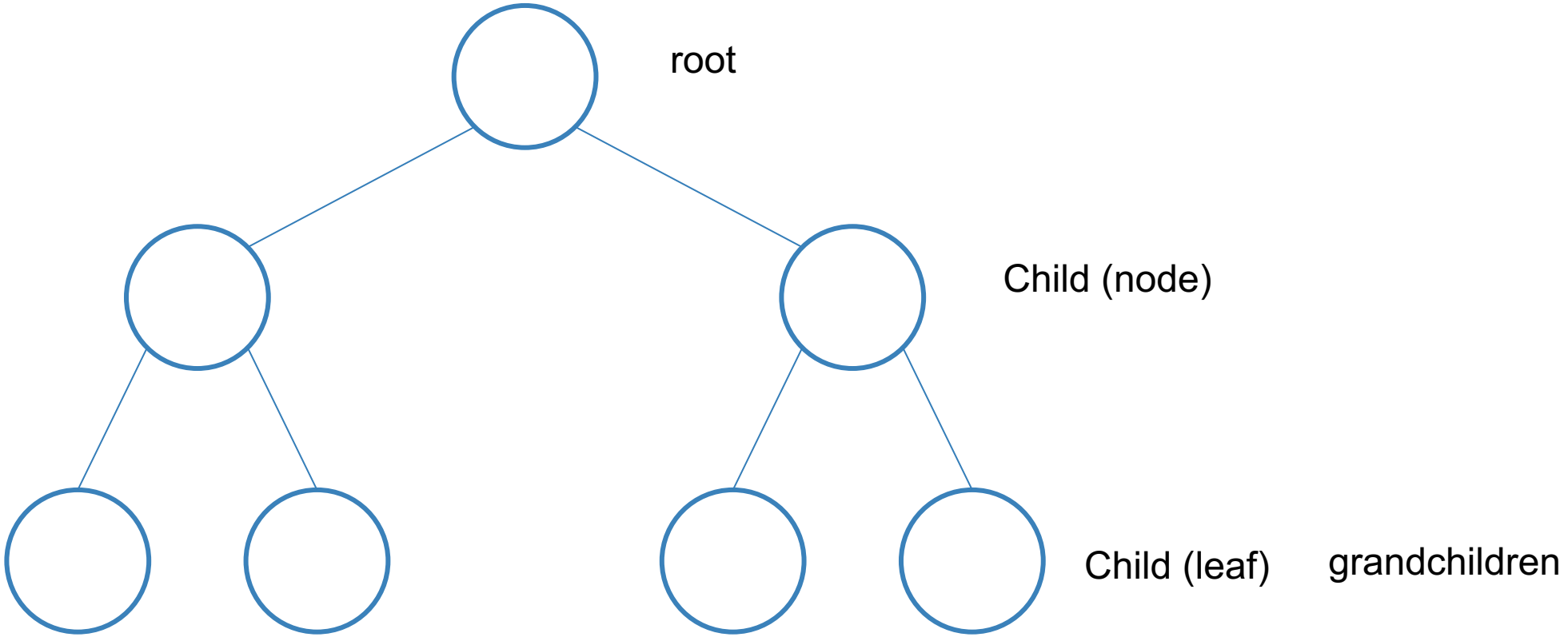
Trees & graph



Tree

- Data structure composed of node
- Each tree has a root node
  - With zero or more “children”
  - Each ”child” has zero or more “children”
- NO CYCLES

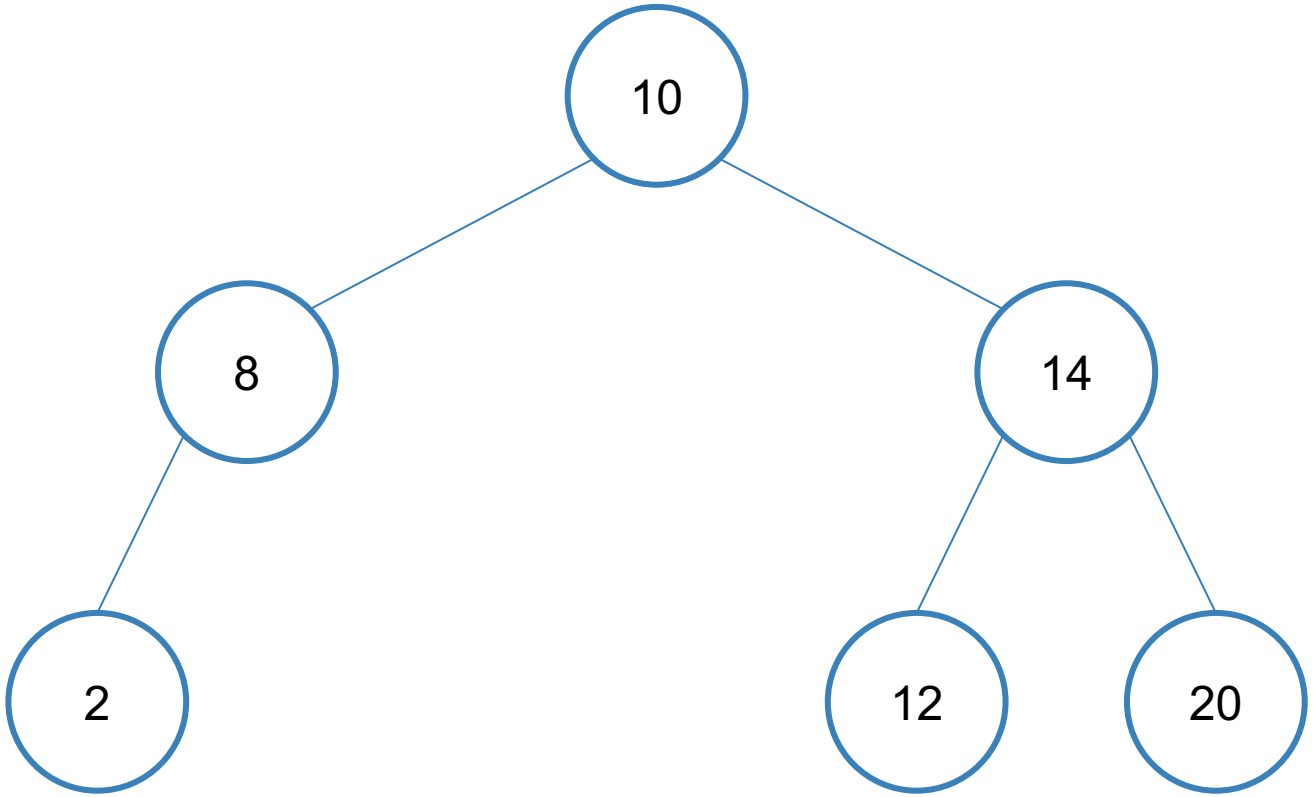
Tree



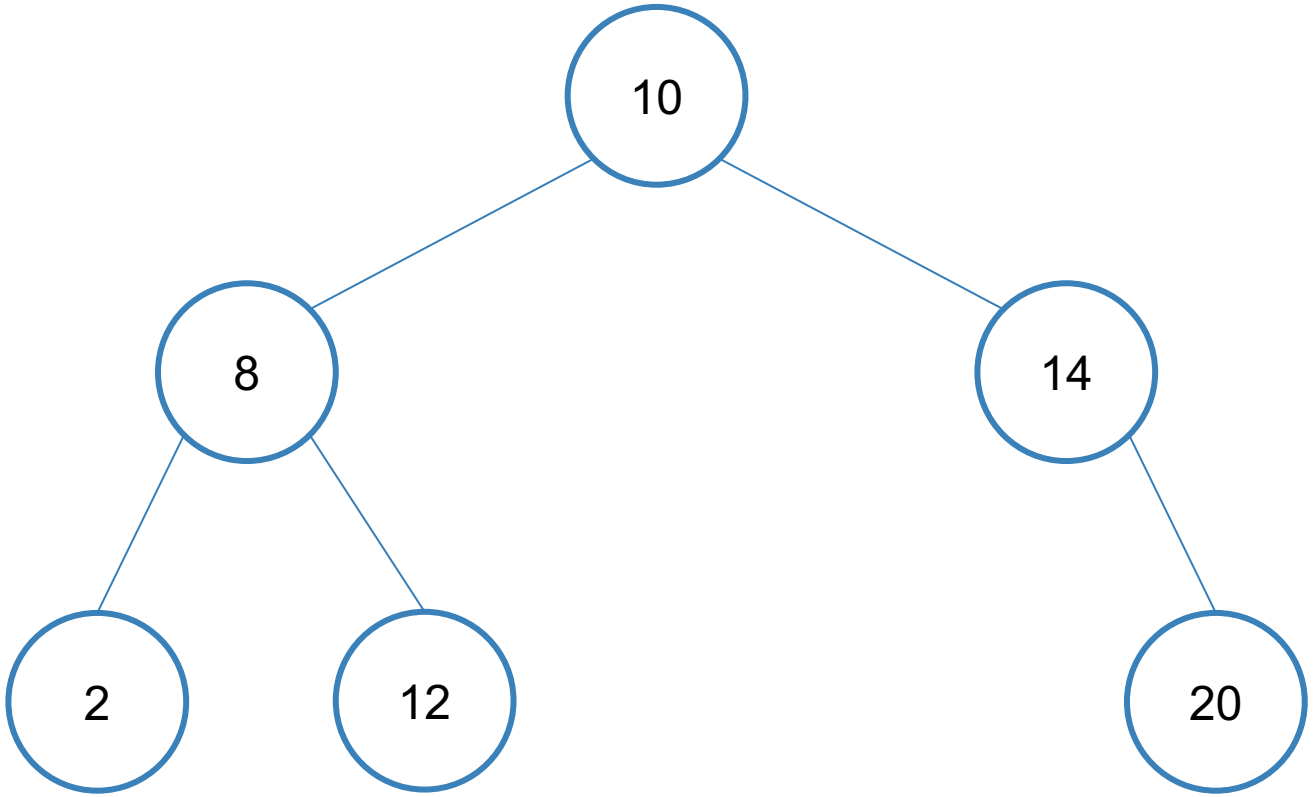
Binary Tree

- Each node has max 2 children
- All left descendants <= n < all right descendants

Binary search



Not a Binary search



Ask

- A binary search tree :
  - for each node, its
    - left descendants <= current node,
    - > right descendants.

Tree balance

- Balanced ~ insert, find in O (log n)
- Not all trees a balance – ask
- Typical a red-black, AVL

Tree walking

- In order traversal: visit left first (ascending)
- Post-order: visit current node after children
- Pre-order: visit current node before children

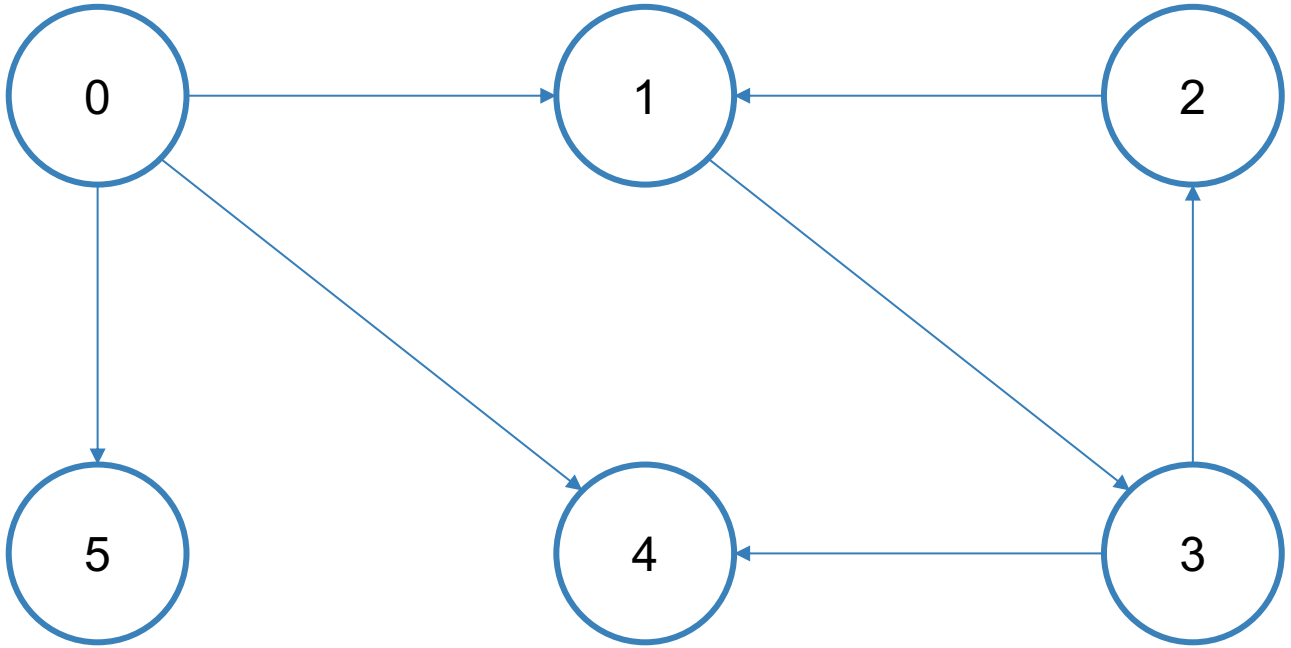
Graphs

- 🤯 A tree is a type of graph
- 🤯 Not all graphs are trees

Graphs

- A tree is a connected graph without a cycle
- A graph is a collection of nodes with edges between them
  - Directed – one way street
  - Undirected – two-way street

Visual



Implementing Graphs

- Adjacency list
  - Every node (or vertex) store a list of adjacent vertices

```
class Graph:
    def __init__(self):
        self.nodes = []

class Node:
    def __init__(self, name, children=[]):
        self.name = name
        self.children = children

    def add(self, node):
        self.children.append(node)
```

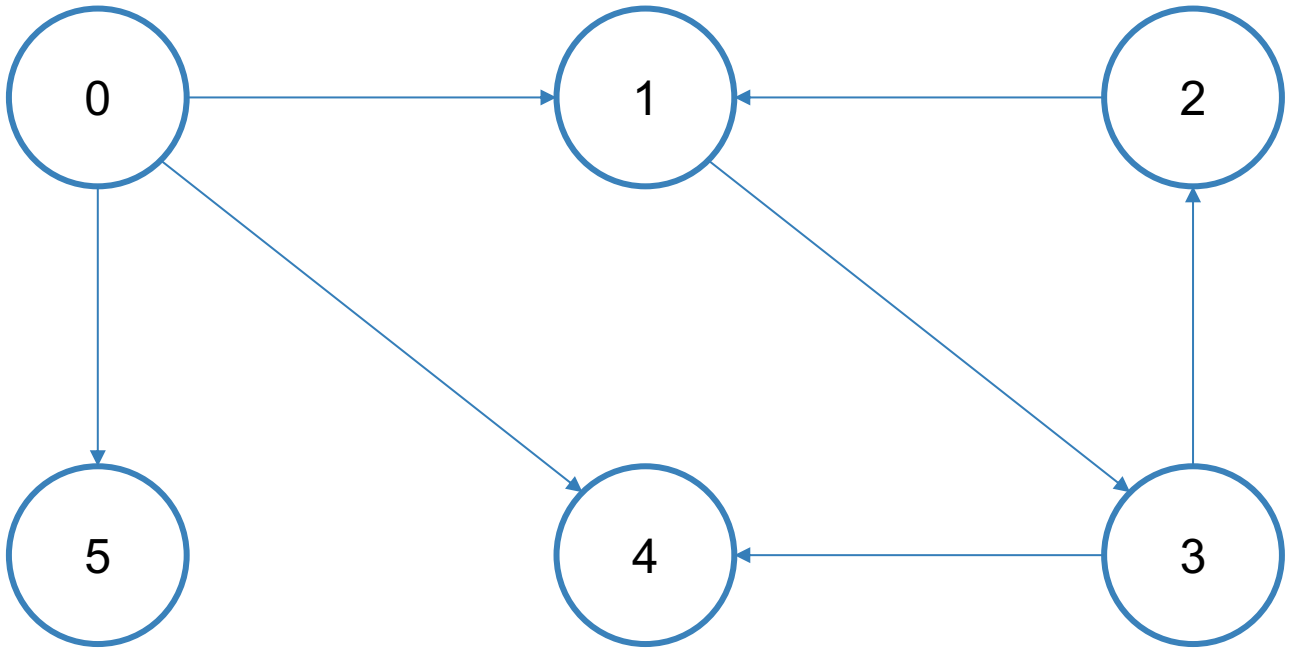
Implementing Graphs

- Adjacency Matrices

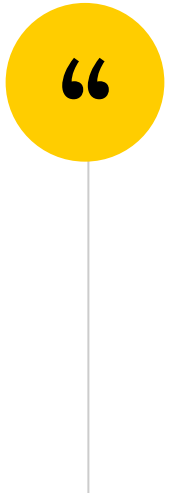
graph[i][j]=true if there is an edge between nodes I & j

Searching

- Depth-First
- Breadth-First
- Bidirectional

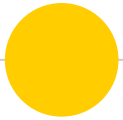


Big O



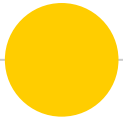
This is super important

- Big O used to describe efficiency of algorithms
- Dedicated courses to the concepts, TAKE THEM
- You need to be able to judge is your algorithm getting faster, slower, do you have enough memory?



Big what?

- Sending file to a friend



Big what?

- Sending file to a friend
- Dropbox? FTP? S3? Plane?

Big what?

- Sending file to a friend
- Dropbox? FTP? S3? Plane?
  - 10 Mbps?

Big what?

- Sending file to a friend
- Dropbox? FTP? S3? Plane?
  - 10 Mbps?
- What if no plane and you drive?

Let's assume

- Sending file to a friend
- Dropbox? FTP? S3? Plane?
  - 10 Mbps?
- What if no plane and you drive?
- Physical delivery time is constant
  - O(1)
- Upload time
  - O(file_size)

Most common Os

- O(1) – one line of code
- O(log n) – divide in half
- O(n log n) – line of code and divide
- O(n) linear – loop
- $O(n^2)$, $O(2^n)$ – nested loop

Jupyter

Notations

- Academics use big 0, big 0 (theta), and big (omega) to describe runtimes
- big 0 describes an upper bound on the time
- big omega for lower bound
- big theta is both -> gives a tight bound
- In MOST interviews theta and big O are merged

A simplification

- Quick sort
  - Best: O(n)
  - Worth: $O(n^{2)}$
  - Expected: O(n log n)
- Any algorithm could run at O(1)

Space complexity

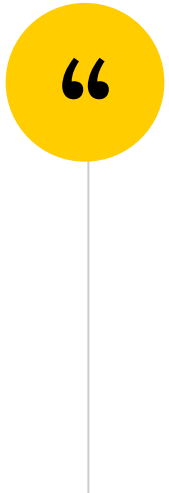
```
def sum(n):
    if n <= 0:
        return 0
    return n + sum(n-1,s)
```

```
sum(5,0)
```

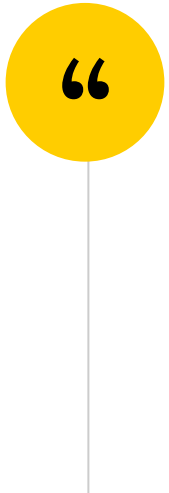
```
sum(5)
  sum(4)
    sum(3)
      sum(2)
        sum(1)
Done with 0

15
```

Lab complexity



Almost there



Prepare for quiz

- Next quiz: June 8, 11:30 AM!
- Data structures: list, array, tree, graph
- Computational complexity

Homework Complex DS-OO

- Plot complexities in one graph
- OO graph
- Test script

The end

- Know your libraries
- Array has types, list is type-less

That's it for today