Python for the courageous

Dealing with technical interviews with the right tools.

Mai Giménez

The cave you fear to enter holds the treasure you seek.

—Campbell

My credentials

- Applied a lot.
- Did quite a few Interviews.
- Passed some.
- Failed some.

The hero's journey

01
Want it

02
Get an ally

03
Train

04
The interview

05
After the interview

Want it

Worthiness doesn't have prerequisites.

—B. Brown

Get yourself ~~an~~ ~~ally~~ a wolf pack







The heroine's journey

01
Want it

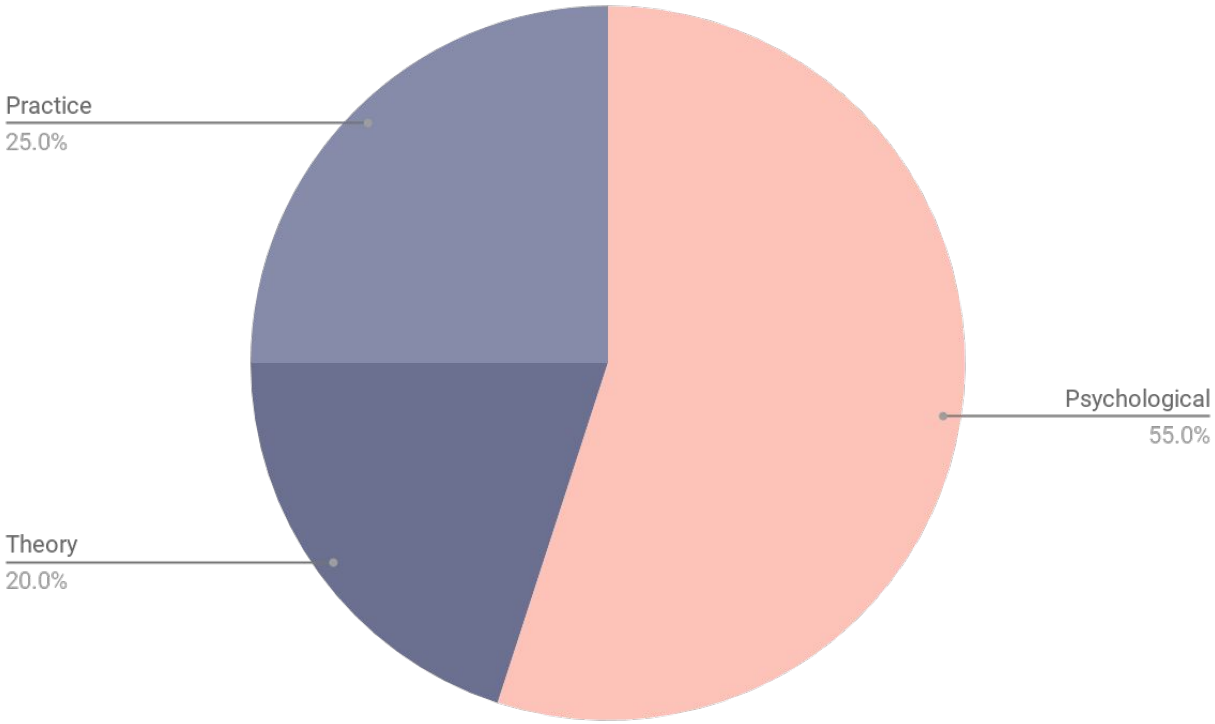
02
Get an ally

03
Train

04
The interview

05
After the interview

Prepare yourself



Build an astonishing CV

Study

01

Data structures

Matrix
Linked list
Hash map
Stack / Queue
Tree / Graph
Heap
Classes
collections

02

Algorithms

Sorting algorithms.
Bisect
Recursivity
Dynamic Programming
`itertools`.

03

Others

Time & Space complexity
How to parallelize
Annotations
Python magic
Speciality

01
Data Structures

Strings

- Immutable sequences of Unicode code points.
- f-strings

```
input_str = ‘Hola Canarias’

input_str.is_alpha()
input_str.is_digit()
input_str.is_space()

input_str.split(sep=None)
input_str.strip([chars])
separator.join(iterable_strs)

input_str.found(sub_str)
```

Python list

- Matrix
- List

- memarray
- Linked list

```
# Matrix
labyrinth = [[False for _ in range(5)]
             for _ in range (5)]

x.append(item)       # Amortized O(1)
x.extend(iterable)   # O(k)
x.insert(pos, item)  # O(n)
x.remove(item)       # O(n)
x.pop(pos)           # O(k)
del x[pos]           # O(k)
x.index(item)        # O(n)
x.count(item)        # O(n)
```

Hash map

- Dictionary from the standard library.
- Data structure that maps keys to values.
- Look-up: O(1)
- Resize when ⅔ full
- collections:
  - *defaultdict*
  - *OrderdedDict*

```
d = {'apples': 1, 'carrots': 2}

d['apples']      # O(1) - O(n)
d['apples'] = 42 # O(1) - O(n)
del d['apples']  # O(1) - O(n)

from operator import itemgetter
sort_by_key = sorted(d.items(),
                     key=itemgetter(0))
sort_by_value = sorted(d.items(),
                       key=itemgetter(1))
```

Queue & Stack

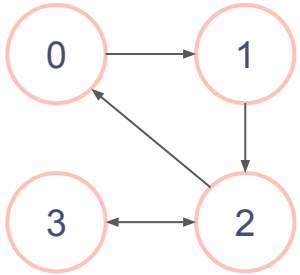
- queue: FIFO data structure
- stack: LIFO data structure

- **deque**: generalization of an stack and a queue

```
# Queue
queue = deque() # O(n)
queue.append(x) # O(1)
queue.popleft() # O(1)
queue[0]        # O(1)

#Stack
stack = []      # O(n)
stack.append(x) # O(1)
stack.pop()     # O(1)
stack[-1]       # O(1)
```

Tree & Graph

- Tree: connected graph without cycles
- Graph: nodes + edges
  - Directed/Undirected
  - Connected/Isolated
  - With/Without cycles




```
# Adjacency list
graph = [[1],[2],[0, 3], [2]]

# Adjacency matrix
graph = [[False, True, False, False],
         [False, False, True, False],
         [True, False, False, True],
         [False, False, True, False]]
```

BFS & DFS

```
def bfs(root):
  queue = deque(root)
  visited = {}
  while queue:
    node = queue.popleft()
    visit(node)
    visited.add(node)
    for n in node.adjacents():
      if n not in visited:
        queue.append(n)
```

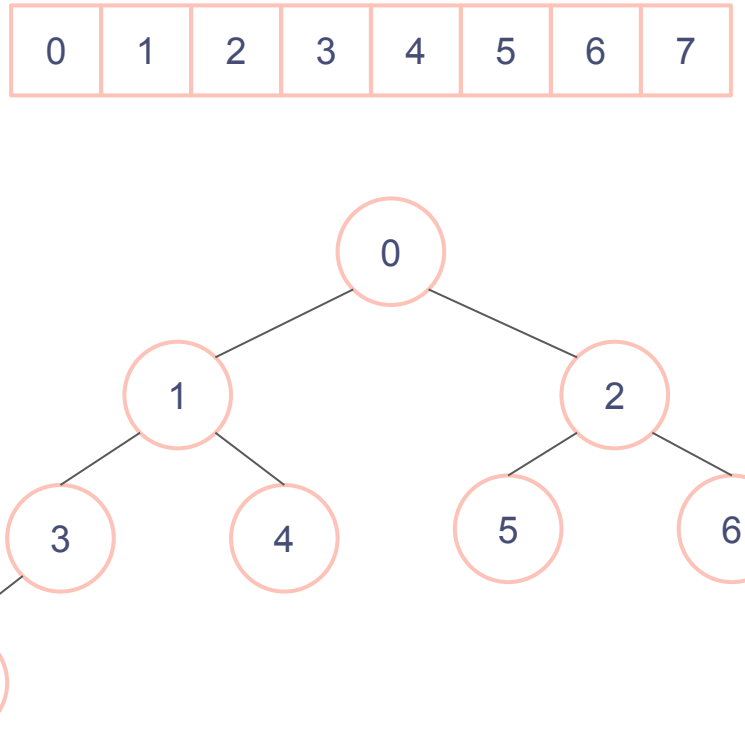
```
def dfs(root, visited):
 if not root: return
 visit(root)
 visited.add(root)
 for node in root.adjacents():
   if node not in visited:
     dfs(node, visited)
```

Heap

- Binary trees where every parent node has a value less than or equal to any of its children.
- Implemented with lists:
  - `heap[k] <= heap[2*k+1]`
  - `heap[k] <= heap[2*k+2]`
- Min-heap. Negative for max-heap.

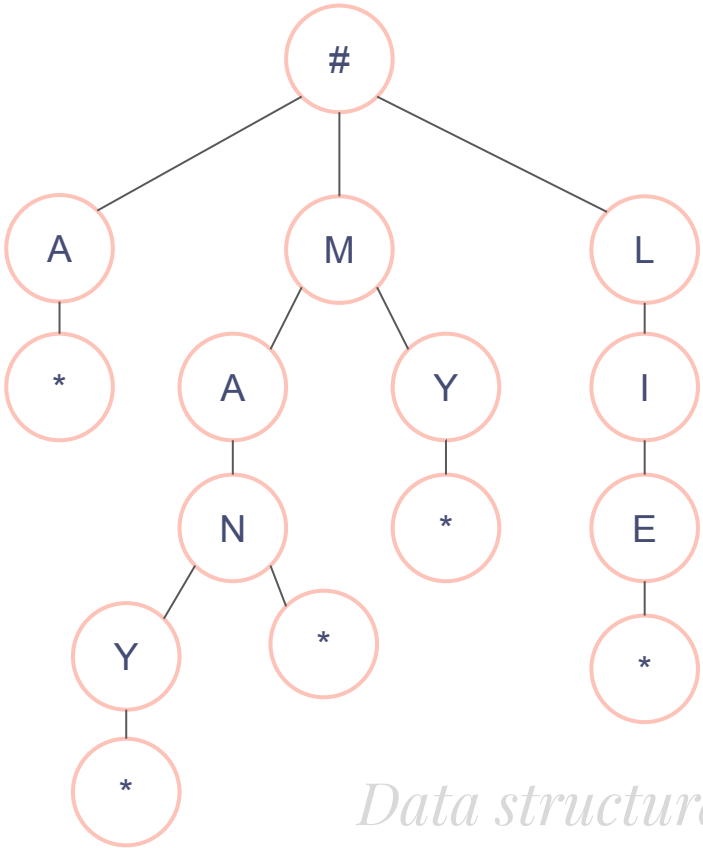
```
heapq.heapify(x)
heapq.heappush(x, item)  # O(log n)
heapq.heappop(x)         # O(log n)
heapq.nsmallest(n, x)

heapq.heappushpop(x, item) # push + pop
heapq.heapreplace(x, item) # pop + push

heapq.merge(*iterables)
```

Heap



Trie

- trie: tree with a character at each node.
- Encodes words with its prefixes
- Validate a word of length k: O(k)



Classes

- Object oriented programming.
- Methods + attributes.

```
class Node:
 def __init__(self):
  self.left = None
  self.right = None

my_tree = Node()
```

collections

- Counter: Dictionary subclass counting hashable objects
- Namedtuple: tuples with named positions
- Defaultdict: subclass of dict with a default factory. (None)

```
# Counter
c = collections.Counter(iterable)
c.most_common([n])
c.elements()
c.update(iterable)

# Namedtuple
Point = namedtuple('Point', ['x', 'y'])

# Defaultdict
d = defaultdict(list)
d[key].append(value)
```

Study

01

Data structures

Matrix
Linked list
Hash map
Stack / Queue
Tree / Graph
Heap
Classes
collections

02

Algorithms

Sorting algorithms.
Bisect
Recursivity
Dynamic Programming
`itertools`.

03

Others

Time & Space complexity
How to parallelize
Annotations
Python magic
Speciality

02
Algorithms

Sorting Algorithms

- Sorting $O(n^2)$:
  - Bubble sort.
  - Selection sort.
  - Insertion sort.

- Efficient sorting
  - Merge sort: O(n log n).
  - Quick sort: O(n log n) - $O(n^2)$.
  - Heapsort: O(n log n)
  - Timsort: O(n log n)

```
new_obj = sorted(items)
new_obj = sorted(items, key=itemgetter(2))
new_obj = sorted(items,
                 key=attrgetter('age'))
items.sort()

def heapsort(items: Iterable[Any]):
  min_heap = heapq.heapify(items)
  items = [heapq.heappop(min_heap)
           for _ in range(len(items))]
```

Bisect

- List in sorted order.
- Sort is mantainted after inserting.

```
# Bisect
bisect.bisect_left(list, item)  # O(log n)
bisect.bisect_right(list, item) # O(log n)

bisect.insort_left(list, item)  # O(n)

bisect.insort_right(list, item) # O(n)
bisect.insort(list, item)       # O(n)
```

Recursivity

- Elegant solution
  - Case base
  - Recursive case

- Space complexity: O(n)
- Time complexity: $O(k^n)$
- Recursive stack > stack

- Iterative with stack.

Recursivity

```
def recursive_fib(n):
 if n == 0 or n == 1:
   return 0
 else:
   return (recursive_fib(n-1) +
           recursive_fib(n-2))
```

```
def fib(n):
 partial_stack = [0, 1]
 for i in range(2, n):
   partial_stack.append(
           partial_stack[-1] +
           partial_stack[-2])
 return partial_stack[n]
```

Dynamic Programming

- Divide an optimisation problem in subproblems.
  - Backtracking.
  - Memoization.
- Store partial solutions.

Dynamic Programming

- Divide an optimisation problem in subproblems
- Store partial solutions.
- Start being **greedy** and optimize
  - Create a candidate(s)
  - Selection function.
  - Feasibility function.
  - Objective function.
  - Solution function.

itertools

- Python batteries
- Fast, memory efficient iterator tools

```
# Infinite iterators
count(start)
cycle(iterable)
repeat(element)

# Shortest chain
chain(*iterables)
dropwhile(pred, iterable)
takewhile(pred, iterable)
islise(iterable, [start], stop, [step])
zip(*iterables)
```

Study

01

Data structures

Matrix
Linked list
Hash map
Stack / Queue
Tree / Graph
Heap
Classes
collections

02

Algorithms

Sorting algorithms.
Bisect
Recursivity
Dynamic Programming
`itertools`.

03

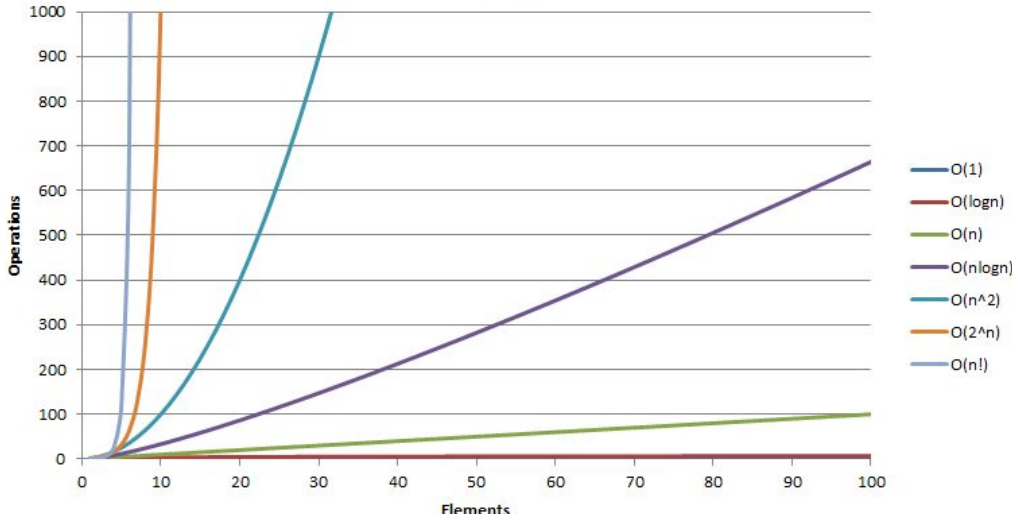
Others

Time & Space complexity
How to parallelize
Annotations
Python magic
Speciality

03
Other

Time & Space Complexity

- Corner cases
- Time complexity:
  - Best
  - Worst
  - Amortized
- Space complexity:



Other

Other

- Scalability
- Annotations
- `functools`
  - @lru_cache
  - partial
- Functional Python:
  - filter
  - map

- Math
- Operating systems
- Your speciality

The hero's journey

01
Want it

02
Get an ally

03
Train

04
The interview

05
After the interview

The interview

Acknowledge your fear,
and do it anyway.

—R.Arzon

The interview

- Be on time
- If you control your breath, you'll control your mind.

The interview

- Be on time
- If you control your breath, you'll control your mind.
- Listen.
- Ask questions.
- Play around with the problem.
- Think about corner cases.

The interview

- Be on time
- If you control your breath, you'll control your mind.
- Listen.
- Ask questions.
- Play around with the problem.
- Think about corner cases.
- Design a modular solution.
- Code.
- **Test**

After the interview

- Acknowledge what you achieved.
- Say thanks.
- Rest.
- Train again.

I did what I knew how to do. Now that I know better, I do better.

—Maya Angelou

Python for the courageous

Mai Giménez
Twitter: ***@maidotgimenez***
Github: ***maigimenez***