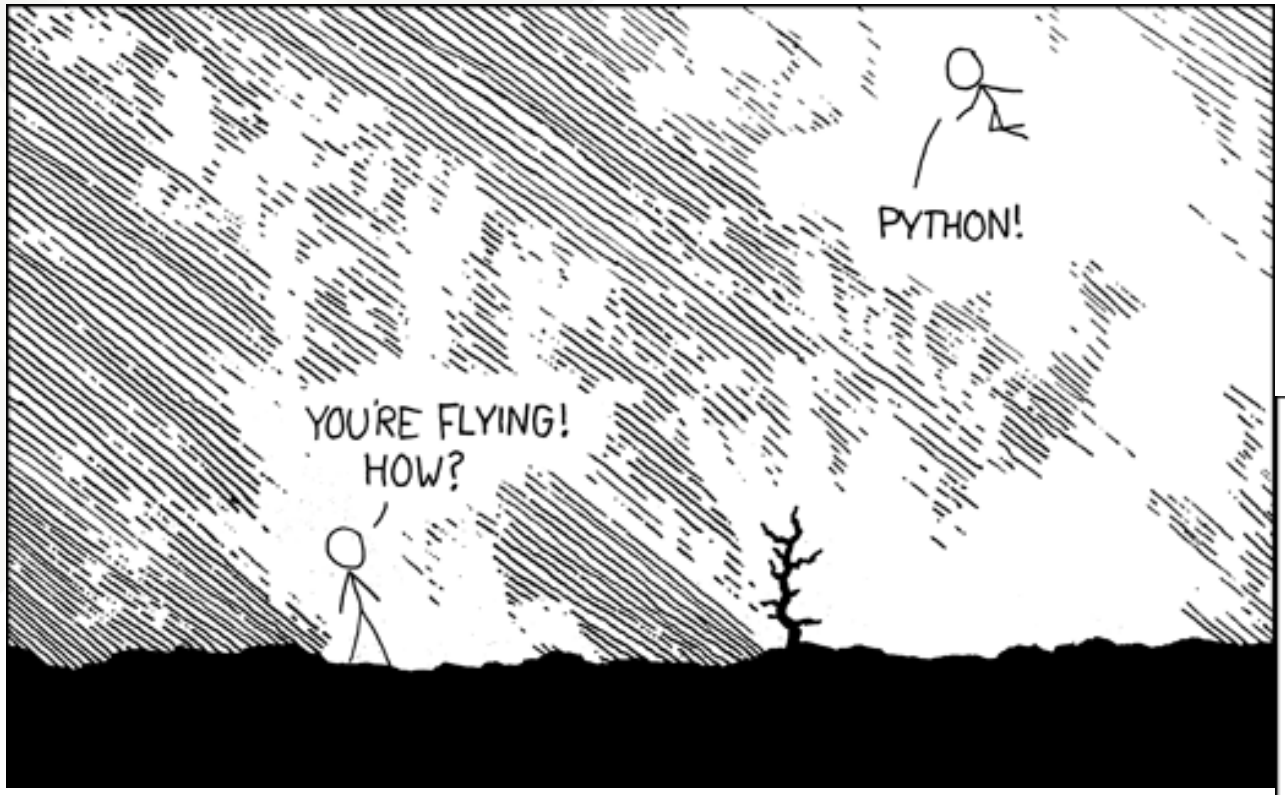
INTRO TO PYTHON

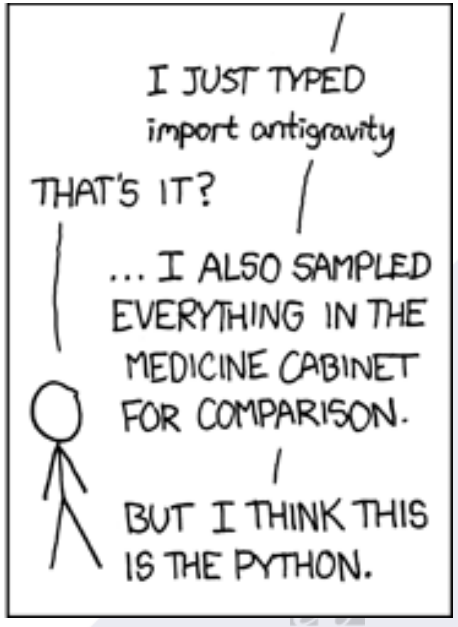
UML ASSOCIATION FOR COMPUTING MACHINERY

Adapted from: https://umlacm.org/assets/resources/2019/fall/PythonPresentation.pdf

UMASS LOWELL

WHAT IS PYTHON?





WHAT IS PYTHON?

- Interpreted
- High-level
- General-Purpose
- Dynamically Typed
- Multi-Paradigm

INTERPRETED

- A.K.A. Scripting Language
- Code Read and Processed at Run-Time
- No Compilation Step Before Processing
- Examples: JavaScript, Perl, Lua, MATLAB

HIGH-LEVEL

- A Loose Term
- Vaguely Implies it does not have direct access to hardware and memory
- Also has the implication that it can be quite slow
- Lots of abstractions

GENERAL-PURPOSE

- This is fairly self-explanatory
- Can Do Anything A ‘Programming Language’ Can
- Does NOT imply it’s great at everything

DYNAMICALLY TYPED

- Variable Type Enforcement does not exist innately
- Casts are, mostly, implied

MULTI-PARADIGM

- There are many ways to approach Python programming
- Functional Programming
- Symbolic Programming (LISP)
- Object Oriented Programming (Like C++, Java)
- Procedural Programming (Like C)

WHAT IS IT USED FOR

- Back End Server Management
- Front End Page Delivery
- Automation and Scripting
- Testing
- Mathematical Proofs
- Data Analytics and Science
- Machine Learning

VARIABLES

- No need to declare just assign
- Can change type after being assigned
- Naming Rules
  - Cannot start with number
  - Variety of Unicode characters supported
  - Case sensitive
  - Cannot be a Python keyword
  - Unlimited Length

TYPES

- Has data types, but does not check them at run time
- Use type() to get the type of a variable at run time
- Compared to C/C++
  - short, int, long, etc… → int
  - float, double, long double → float
  - char, char[] → str
  - bool → bool (True, False)
  - void/NULL → None

TYPING

- Can optionally mark types of variables, parameters, return values, and members
- Use a colon after the name: e.g. x: int = 9
- This does not prevent you from using other types for this variable
- As of 3.10 can combine multiple types with ‘|’: x: float|int = 9

SYNTAX

- Significant whitespace
- Comments indicated with ‘#’
- Loops, if, etc… do not use parenthesis
- Curly braces are used for sets and dicts, not code blocks
- elif used instead of else if
- for loops have different format
  - for i in range(5):

OUTPUT

- print()
- Parameters
  - *objects : items to print, uses __str__ or __repr__
  - sep : separator between each item
  - end : character(s) at end of line
  - file : file to print to, use None to print to stdout
  - flush : force stream flush

INPUT

- input()
- Optional string parameter for prompt
- Returns value entered as string

FUNCTIONS

- Declared using def keyword
- No Overloading
- Same naming rules as variable names
- Can assign default values to parameters
- Example:

```
def my_func(x, y, z="python"):
    return x + y + z
```

LAMBDAS

- Use lambda keyword
- Typically one line
- Examples
  - f = lambda: print(“Hello World”)
  - s = lambda a,b,c: print(a, b, a, c, a)
  - t = lambda a, b: a * b + b * a
- For multiple line of code use a function

CLASSES

- Declared with class keyword
- Member functions have first parameter as self
- One constructor, __init__(self)
- Do no declare variables in class body, instead do so in __init__
- No writing getters and setter, just use the members directly

CLASSES

- Example:

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

    def as_list(self):
        return [self.x, self.y]
```

COLLECTIONS

List	[w, x, y, z]
Dict	{w:x, y:z}
Set	{w, x, y, z}
Tuple	(w, x, y, z)

COMPREHENSIONS

- Quicker and more efficient way to construct collections
- Examples:
  - Squares = { i:i*i for i in range(13) }
  - Multiplication_table = [[i*j for j in range(13)] for i in range(13)]
  - Even_and_threeven = [i for i in range(101) if i%2==0 and i%3==0]

EXEC & EVAL

- Warning! Here be danger
- Eval takes a string, evaluates it as a Python expression, and returns the result
- Exec is similar, but it is designed to take multiple line of code and execute it

WHERE TO CODE?

HELPFUL LIBRARIES

- math & cmath
- re
- collections
- random
- pathlib
- itertools
- functools
- dataclasses
- os & sys
- typing & collections.abc
- string
- json & html & xml
- datetime
- decimal & fraction
- subprocess
- antigravity

ZEN OF PYTHON

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Flat is better than nested.
- Sparse is better than dense.
- Readability counts.
- Special cases aren't special enough to break the rules.
- Although practicality beats purity.
- Errors should never pass silently.
- Unless explicitly silenced.
- In the face of ambiguity, refuse the temptation to guess.
- There should be one-- and preferably only one --obvious way to do it.
- Although that way may not be obvious at first unless you're Dutch.
- Now is better than never.
- Although never is often better than *right* now.
- If the implementation is hard to explain, it's a bad idea.
- If the implementation is easy to explain, it may be a good idea.
- Namespaces are one honking great idea -- let's do more of those!

TECHNICAL INTERVIEWING

UML ASSOCIATION FOR COMPUTING MACHINERY

Adapted from: https://shorturl.at/WayEa

UMASS LOWELL

WHAT IS TECHNICAL INTERVIEWING?

- Most companies will ask you to demonstrate your knowledge during an interview
  - Usually, the employer will ask you to solve these coding problems through **Online Assessments,** or **OAs**
- OAs come in one of three forms:
  - HackerRank Assessment
  - Live coding with an employee from the company
  - Both; Some companies will do both assessments if the position you applied to has multiple interviewing rounds
- Companies are **NOT** looking for the correct answer in your OA, but rather assessing your technical and problem-solving skills

WHAT CONCEPTS ARE COVERED IN AN OA?

Time Complexity

- Unit of measurement for the running time of the algorithm
- Time and space

Data Structures and Algorithms

- Data Structures:
  - Arrays and Hash Tables
  - Linked Lists
  - Trees
- Algorithms:
  - Binary Search
  - Quicksort
  - Tree Traversal
- Recursion

Object-Oriented Programming

- Classes and Objects
- Four Principles:
  - Abstraction
  - Encapsulation
  - Inheritance
  - Polymorphism

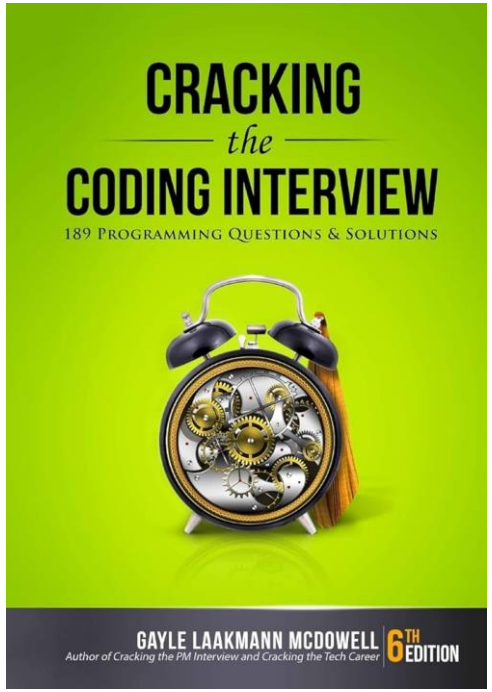
Understanding the concepts is more important than finding the right answer!

TIPS AND TRICKS FOR TECHNICAL INTERVIEWS

- If you get stuck:
  - Try another way to solve the problem
  - Ask for a hint
- Code out loud!
  - It's important for the interviewer to gauge your understanding of the problem
  - It also allows you to think about the problem
- Practice!
  - You can either grind 200 LeetCode problems or understand the patterns and concepts behind each problem; the choice is yours!

RESOURCES

- LeetCode
  - Largest collection of online practice problems
  - Questions on algorithms and much more
  - LeetCode Premium: Exclusive problems from top companies such as Amazon and Google
- HackerRank
  - Most of your OAs will come from here
  - Large amount of practice problems
- Cracking the Coding Interview
- YouTube: Neetcode



SAMPLE PROBLEMS

UML ASSOCIATION FOR COMPUTING MACHINERY

Adapted from: https://shorturl.at/WayEa

PROBLEM 1 – SQUARES OF A SORTED ARRAY

Given an integer array nums sorted in **non-decreasing** order, return *an array of **the squares of each number** sorted in non-decreasing order*.

Example 1:
Input: nums = [-4,-1,0,3,10]
Output: [0,1,9,16,100]

Example 2:
Input: nums = [-7,-3,2,3,11]
Output: [4,9,9,49,121]

PROBLEM 2 – SINGLE ELEMENT IN A SORTED ARRAY

You are given a sorted array consisting of only integers where every element appears exactly twice, except for one element which appears exactly once. Return *the single element that appears only once*. Your solution must run in O(log n) time and O(1) space.

Example 1:

Input: nums = [1,1,2,3,3,4,4,8,8]

Output: 2

Example 2:

Input: nums = [3,3,7,7,10,11,11]

Output: 10

PROBLEM 3 – MERGE TWO SORTED LISTS

You are given the heads of two sorted linked lists list1 and list2. Merge the two lists into one **sorted** list. The list should be made by splicing together the nodes of the first two lists. Return *the head of the merged linked list*.

Example 1:
Input: list1 = [1,2,4], list2 = [1,3,4]
Output: [1,1,2,3,4,4]

Example 2:
Input: list1 = [], list2 = []
Output: []

Example 3:
Input: list1 = [], list2 = [0]
Output: [0]

PROBLEM 4 – TWO SUM

Given an array of integers nums and an integer target, return *indices of the two numbers such that they add up to target*. You may assume that each input would have ***exactly*** **one solution**, and you may not use the *same* element twice. You can return the answer in any order.

Example 1:
Input: nums = [2,7,11,15], target = 9
Output: [0,1]

Example 2:
Input: nums = [3,2,4], target = 6
Output: [1,2]

Example 3:
Input: nums = [3,3], target = 6
Output: [0,1]

MORE PROBLEMS

- Super Easy Problems (for beginners): https://leetcode.com/problem-list/prshgx6i/
- Top Interview 150: https://leetcode.com/studyplan/top-interview-150/
- Leetcode 75: https://leetcode.com/studyplan/leetcode-75/

MORE PROBLEMS

- Super Easy Problems (for beginners): https://leetcode.com/problem-list/prshgx6i/
- Top Interview 150: https://leetcode.com/studyplan/top-interview-150/
- Leetcode 75: https://leetcode.com/studyplan/leetcode-75/

MORE PROBLEMS

- Group 1 (Easy):
  - Squares of a Sorted Array
  - Two Sum
  - Fizz Buzz
  - Valid Parentheses

- Group 2 (Medium to Hard):
  - Longest Palindrome Substring
  - 4 Sum
  - Plus One
  - Merge Two Sorted Lists

CONTACT INFO

Our Linktree!