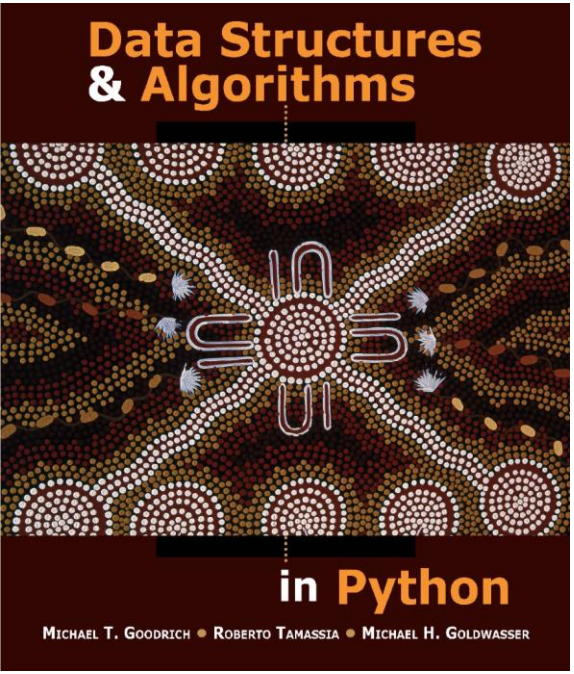
CS-UY 1134

Data Structures & Algorithms

Professor: Phyllis Frankl
Email: pfrankl@nyu.edu

Office Hours: Wednesday 6pm-7pm +
By appointment



"Data Structures & Algorithms in in Python, 1st Edition, Goodrich, Tamassia, Goldwasser

Final Grade Breakdown:	20%	First midterm exam
	20%	Second midterm exam
	30%	Final exam
	20%	Homework
	10%	Lab Grade

Python Version:	3.x

IDE:	PyCharm
	IDLE

Overview of the Course

- Knowledge of Data Structures and Algorithms starts to mark the difference between a computer scientist and a programmer:
  - Understanding of how to evaluate efficiency of different approaches to solving a problem
  - Toolbox of useful data structures (ways of organizing and accessing data) that are useful for solving many different problems
  - Basic algorithms for frequently encountered problems: sorting, searching, working with graphs

Basic Data Structures

- Stacks: sequence with last-in-first-out behavior
  - Applications to programming language compilation and execution and evaluating expressions
- Queues: sequence with first-in-first out behavior
  - Applications to problems where many actors want to access a resource – networks, operating systems, simulations of real world, …

More basic data structures

- Trees: Represent Hierarchies; applications to Searching and sorting, representing dictionaries, organizing file systems, …
- Graphs: Represent more general relationships between objects; applications to communication and transportation networks, social networks, ...

Data

- Build-in types
  - int
  - float
  - bool
  - str
  - list
  - tuple
  - dict
  - …
  - …
- Programmer defined types (classes)

Expressions

- I/O expressions
- Assignment
- Arithmetic expressions
- Boolean expressions

Control Flow

- Sequential
- Branching
  - if
  - if-else
  - if-elif-else
- Iterative
  - while
  - for
- Function calls
- Exceptions

Mutation vs. Construction

Mutating the List	Constructing New Lists
indexed assignment (lst[i] = …)	list literals ([1, 2, 3], [ ])
append method	list constructor (list(…))
insert method	+ operator
pop method	* operator
reverse method	slicing (lst[__:__])
sort method	copy.copy function
extend method	copy.deepcopy function
+= operator	list comprehension