Data Structures

Spring 2026

Tony Stone

Contents

1 Preface

1.1 Mission Statement

With this work, I set out to develop a comprehensive set of lecture notes, on the topic of Data Structures. While developing these notes, I am keeping in mind both the lecturer, and the learner. My goal is that this will amount to an educational resource that acts as a guide throughout fundamental topics, in a manner which is accessible and painless for all parties. This work should provide confidence and direction to its reader, whether they are at the front of a classroom, or at their dining-room table.

1.2 Acknowledgments

I would like to thank the Center for Academic Success and Accessibilty Services at Southern Connecticut State University, for the employment and opportunity to develop these materials. I would like to extend this thank you for all of the administrative support I have recieved to my boss Kathleen De Oliveira, and coordinaters Tanisha Guadalupe De Jésus, Calleigh Cotter, and Mia Grella. Of course, I would like to extend this thank you to Professor Carl Haberfeld, for allowing me to attend his CSC-212 lectures.

2 Introduction's Introduction

I am very excited to dive into the wonderful world of **data structures**. Before we get started, there is just a bit of housekeeping to do. These matters are more relevant to the themes of this work, and the text as it appears on paper. I figure this is not really an introduction to data structures, its more of a introduction to data structures's introduction!

2.1 Python Examples

Topics and key ideas will usually correspond to some kind of Python example. An assumption is being made that the reader is to some extent *aquainted* with Python. There is absolutely no need to be a wizard of Pythonism, so there are notes clarifying syntax where necessary.

2.1.1 I Hate Snake Case

Per the title of this soap-box, I am not a fan of 'snake_case' as a naming convention. For the Python examples I will be writing in 'camelCase.' Is it my early Java Indoctrination peaking through? Maybe—but a set naming convention is almost entirely superfluous, and it looks nicer to my eyes. camelCase it is.

Let's get accustomed to it, here's an Example!

(Example01.py)

```python
def sayHi():
    print("Hello world!")

sayHi()
```

The reference to the actual script 'Example01.py' is included for your convenience, it can be found in the corresponding GitHub Repository!

2.1.2 Further Reading

Many of the terms and concepts featured in here will be based on the textbook *Problem Solving with Algorithms and Data Structures using Python* by Brad Miller and David Ranum. They have a fantastic online resource, so please check it out!

- https://runestone.academy/ns/books/published/pythonds/index.html

3 Introduction to Data Structures

3.1 Defining Data

Let's begin with getting the terms out of the way, and associating them with a defintion.

- **Atomic data types**: A datatype which may not be broken down into smaller parts.
- **Abstract data type** (ADT): A logical, abstract definition of how data is to be viewed.
-
- **Primative data type**: A definition of data, with regards to allowable values, and operations.
- **Data structure**: The actual implementation of an ADT in a given programming language.

These terms at first glance seem like sticking 'data' next to a bunch of other computer-y words. Some of their coonnections and relations can be muddy. So let's take a moment to try to understand them through some explanation, and examples.

3.1.1 Atomic Data Types

Atomic data types are types such as *integers*, *floating point numbers*, and *booleans*. Since these types do not get broken down into smaller pieces of data. Note, this is somewhat language-dependent. For example, Python treats it's boolean type bool as a subclass of int, and therefore it is not internally atomic.

3.1.2 Abstract Data Types

Abstract data types are instead defined by *behavior*, what it does. We will look into a data structure called a **Stack**. For now, know that this is a collection, and we define it as a collection which releases objects in reverse-order of their arrival. Note: I did not say this was a collection of any kind of thing, or had any specific operations. For ADTs, we're concerned with how they act!

3.1.3 Primative Data Types

The **primative data types** include data such as *integers*, *floating point numbers* and *booleans*. As given in the above definition, they are defined by the possible values and the possible operations they can participate in. These are used both on their own, but also in the further creation of data structures.

3.1.4 Data Structures

Data structures are the real deal. A data structure is the actual implementation of an ADT which we will interface with. Back to our stack, we might use a standard python list, and only append and pop items. This is a very basic example, with some data-structures later, we will be giving it more effort than this!

But with those definitions, lits now take a look at a python script, and identify what-is-what:

(Example02.py)

```python
# Primative Data Type
print("Im an integer, called ", type(6))

# Abstract Data Type
print("I want to release data in reverse order of it's
    arrival")

# Data Structure
stack = [1, 2, 3]
stack.append(4)
print(f"4 comes and goes: {stack.pop()}, leaving {stack}")
```

- An integer or floating point number are primative data types.
- ADTs are not the implementation, just the thoughts.
- We can implement a stack with a list, and treat it like one.

3.2 Moving Forward

I will use these terms accordingly. For basic components, I'll say "data type." "ADT" will be used when discussing the abstract way we're thinking of different kinds of data. "Data type/data structure" will be used when referencing a specific implementation, in python or otherwise. I will also use "data structure" to talk about the subject as a whole.

4 Basic Data Types and Structures

4.1 Integer, Floating Point Numbers, Booleans

4.1.1 Integer

The ADT integer can take a value from the set :$\{-n, ..., 1, 0, 1, ..., n\}$. Almost certainly your machine cannot hold the highest conceivable n, but remember: we don't care about that, were thinking in the abstract!

Anyway, along side those values, we can imagine the kind of operations that get performed on integers. We add them, subtract them, multiply them, and perform *integer division* and *modulo division*. I did not skip over "division" since we can't always get away with it in the abstract world. $2 \div 3$ is a no go! Let's investigate this further:

(`Example03.py`)

```
numOne = 1
numTwo = 2

#Check Types
print(type(numOne))
print(type(numTwo))

print(type(numOne + numTwo)) # Addition
print(type(numOne - numTwo)) # Subtraction
print(type(numOne * numTwo)) # Multiplication
print(type(numOne // numTwo)) # Integer division
print(type(numOne % numTwo)) # Modulo division

print(type(numOne / numTwo)) # Devision XX
```

Run this script, and notice that after all of these operations, they are all integers, (Python's 'int' data type). We'll need something a little more robust.

4.1.2 Floating Point Number

The floating point ADT is a number that has an integer part, as well as a fractional part. The type of values a floating point may hold is now: $-n, n$. Once again, this is not with regard to any implementation, so we define this interval as continuous. Actual computer numbers are not continuous, famously.

Feel free to change the initial variables numOne and numTwo to be Python's 'float' type, to get a sense of the operations. All of them are the same, with the added benefit of addition.

4.1.3 Boolean

Booleans as an ADT can take the value of either "*True*" or "*False*." The kind of operations that they can undergo are taken directly from boolean logic (naturally). This includes *AND*, *OR*, and *NOT*. I will spare the details of logical operations here, more information on logic can be found in another one of my lecture notes on Computer Systems.

4.2 Collection Types

We'll keep this conversation in the abstract, without getting deep into any one implementation of these data structures.

4.2.1 Strings

We'll think of a string as a collection of characters. these characters are ordered in so far as the way they are written. We can compare strings, splice strings, concatenate them, etc.

4.2.2 Lists

Lists are like an even more abstract version of strings. While they can be ordered, they definitely don't need to be.

Lists can be indexed, they can be spliced. You can add things and remove things from lists. Tt does not truly matter *what* you store inside of them! We'll dive into the types of lists later on.

4.2.3 Maps

Maps or **Dictionaries** (it is not worth getting stuck in the mud of one language's keyword) are collection types defined by a key-value pair.

Think about it this way, when you go to the bank, and enter your pin number to access your count, what does the bank say:

1. "Yup, definitely your pin number!"

2. "Ah, here's your account information!"

The second one, but why? Well, (for simplicities sake) your pin number is a kind of key that lets the bank look up your account information, and give it to you. If we didn't have this way to map information to each other, we'd have a lot more replies like the first response.

4.2.4 Miscellaneous

We also have **sets** and **tuples**. I do not see these being potent topics for this course, so I will let mentioning them suffice. I encourage you to look further into them!

4.3 Moving forward

We've covered probabily the most encountered ones, with enough detail that will get accross the broad themes (hopefully mostly a reminder from your introductory course on programming). Many of the remaining collection ADTs warrant their own Section!

5 Analysis

References

[1] Bradley N. Miller and David L. Ranum. *Problem Solving with Algorithms and Data Structures Using Python*. Franklin, Beedle & Associates, Wilsonville, OR, 2 edition, 2011.