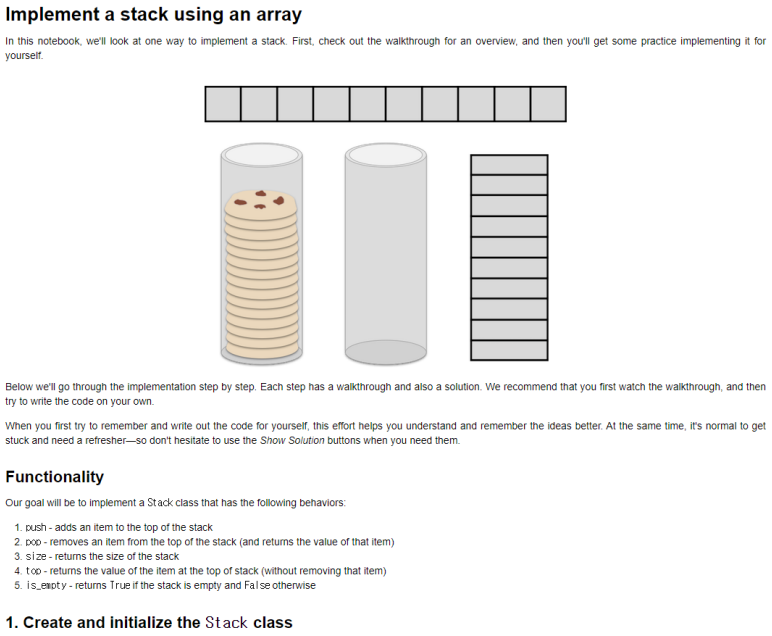
Data Structure using Python

1. https://runestone.academy/runestone/books/published/pythonds/index.html
   - Open source project and 3rd edition
   - Used by several universities
   - Well organized and good explanation (maybe too verbose)
   - Self-check exercise provided every session
   - Not many exercise problems
2. https://github.com/lpinzari/udacity-dsa-nand
   - 설명: **Udacity**라는 **온라인 코스 수업**에 사용된 자료입니다. .ipynb, .py, .md 파일로 동일 내용 자료들이 모두 저장 되어 있으며 설명과 그림도 적절히 나와있습니다. 다만, 정규 대학 강의보단 온라인 코스에 더 가까운 자료인 것 같습니다. (이론에 대한 설명 파일 조금 + 실습 문제 파일 다수)

     각 Chapter별로 Python 기초, Data Structure, Algorithm, Jupyter Notebook에 대해 다루고 있어서 Data Structure 외에도 Python 기초와 Jupyter Notebook 사용법 등의 자료도 사용할 수 있을 것 같습니다. (모인활 수업에 사용하신 자료가 이미 있으시니 참고로만 두어도 좋을 것 같습니다.)
   - 예시:



Implement a stack using an array

In this notebook, we'll look at one way to implement a stack. First, check out the walkthrough for an overview, and then you'll get some practice implementing it for yourself.

Below we'll go through the implementation step by step. Each step has a walkthrough and also a solution. We recommend that you first watch the walkthrough, and then try to write the code on your own.

When you first try to remember and write out the code for yourself, this effort helps you understand and remember the ideas better. At the same time, it's normal to get stuck and need a refresher—so don't hesitate to use the *Show Solution* buttons when you need them.

Functionality

Our goal will be to implement a `Stack` class that has the following behaviors:

1. `push` - adds an item to the top of the stack
2. `pop` - removes an item from the top of the stack (and returns the value of that item)
3. `size` - returns the size of the stack
4. `top` - returns the value of the item at the top of stack (without removing that item)
5. `is_empty` - returns `True` if the stack is empty and `False` otherwise

1. Create and initialize the `Stack` class

1. Create and initialize the `Stack` class

First, have a look at the walkthrough:

In [ ]:

In the cell below:

- Define a class named `Stack` and add the `__init__` method
- Initialize the `arr` attribute with an array containing 10 elements, like this: `[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]`
- Initialize the `next_index` attribute
- Initialize the `num_elements` attribute

In [1]:
```
class Stack:

    def __init__(self, initial_size = 10):
        self.arr = [0 for _ in range(initial_size)]
        self.next_index = 0
        self.num_elements = 0
```

Let's check that the array is being initialized correctly. We can create a Stack object and access the `arr` attribute, and we should see our ten-element array:

In [2]:
```
foo = Stack()
print(foo.arr)
print("Pass" if foo.arr == [0, 0, 0, 0, 0, 0, 0, 0, 0, 0] else "Fail")
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Pass
```

In [ ]:
```
class Stack:

    def __init__(self, initial_size = 10):
        self.arr = [0 for _ in range(initial_size)]
        self.next_index = 0
        self.num_elements = 0
```

2. Add the `push` method

Next, we need to define our push method, so that we have a way of adding elements to the top of the stack.

In [ ]:

Now give it a try for yourself. Here's are the key things to include:

- The method will need to have a parameter for the value that you want to push
- Remember that `next_index` will have the index for where the value should be added
- Once you've added the value, you'll want to increment both `next_index` and `num_elements`

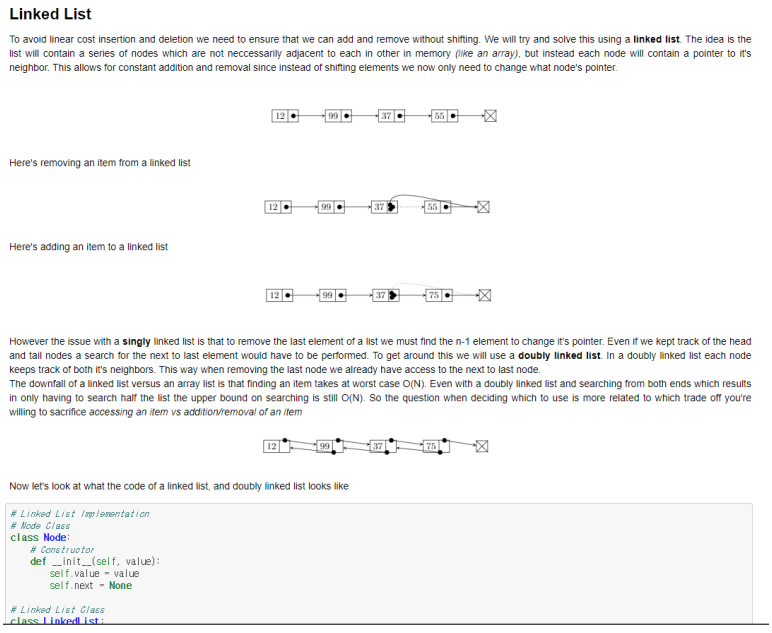
3. https://github.com/gousiosg/algo-ds

   - 설명: 지난번 보내 드렸던 두번째 자료 (https://gousios.org/courses/algo-ds/ - Jupyter notebook으로 옮기면 유용할 것 같다고 말씀하신 자료)의 깃헙 사이트입니다. 다만, Jupyter notebook이 아닌 **Rmarkdown**으로 작성되어 있습니다. 제가 여러 방법으로 .Rmd 파일을 ipynb으로 변환하는 것 까진 성공하였지만 그 Contents 중 context는 괜찮지만 코드에 해당하는 부분들은 오류가 있어 보입니다. 제가 R과 Rmarkdown에는 경험이 없어서 어느 정도까지 활용할 수 있을지 정확히 모르겠습니다..

4. https://github.com/Satokaheni/Data-Structures

   - 설명: University of Texas Dallas에서 사용된 자료입니다. 특정 Chapter를 제외하면 다소 강의에 초점이 맞춰진 자료이여서 많은 설명이 포함되어 있고 실습이 매우 적습니다. 실습 문제나 중간중간 학생들이 직접 프로그래밍 할 수 있도록 바꾸는 것은 다른 자료들에서 가져와서 삽입해도 좋을 것 같습니다.

   - 예시:



Linked List

To avoid linear cost insertion and deletion we need to ensure that we can add and remove without shifting. We will try and solve this using a **linked list**. The idea is the list will contain a series of nodes which are not neccessarily adjacent to each in other in memory *(like an array)*, but instead each node will contain a pointer to it's neighbor. This allows for constant addition and removal since instead of shifting elements we now only need to change what node's pointer.

Here's removing an item from a linked list

Here's adding an item to a linked list

However the issue with a **singly** linked list is that to remove the last element of a list we must find the n-1 element to change it's pointer. Even if we kept track of the head and tail nodes a search for the next to last element would have to be performed. To get around this we will use a **doubly linked list**. In a doubly linked list each node keeps track of both it's neighbors. This way when removing the last node we already have access to the next to last node.
The downfall of a linked list versus an array list is that finding an item takes at worst case O(N). Even with a doubly linked list and searching from both ends which results in only having to search half the list the upper bound on searching is still O(N). So the question when deciding which to use is more related to which trade off you're willing to sacrifice *accessing an item vs addition/removal of an item*

Now let's look at what the code of a linked list, and doubly linked list looks like

```
# Linked List Implementation
# Node Class
class Node:
    # Constructor
    def __init__(self, value):
        self.value = value
        self.next = None

# Linked List Class
class LinkedList:
```

5. https://www.gousios.gr/courses/algo-ds/

   - 설명:

모 기관(대학?)에서 만들고 사용한 자료인 것 같습니다. Course들이 전부 나와있고 해당 Course를 클릭하면 설명과 함께 자세한 예시들이 Jupyter Notebook 형식으로 작성 되어 있습니다. 단, 해당 화면은 편집(수정) 불가능합니다. 또한, 과제가 포함 되어있는데, 과제

에 해당하는 것은 파일로 다운받아 작성하고 제출하는 방식이라 이는 편집(수정) 가능한 jupyter notebook 파일입니다.

- 예시:

Course contents를 포함한 첫 페이지

Contents

Week	Lecture	Topic	Lecturer	Assignment (Deadline)
1	1	Course introduction, Recursion	GG	
1	1	Algorithm Analysis	GG	
2	1	Arrays, Queues and Stacks	MK	Doubly-linked lists (jypyter, html, solutions) (28/11/2017)
2	2	Lists, Sets	MK	
3	1	Trees: basic concepts and binary Trees	JH	Red-Black Trees (jupyter, html, solutions) (12/12/2017)
3	2	More Trees: AVL and B+ Trees	JH	
4	1	Graphs (Representation and Traversal)	PK	Graph algorithms (jupyter, html, solutions) (9/01/2018)
4	2	Graph algorithms (Shortest paths, Topological sorting)	PK	
5	1	Sorting	JH	Searching (jupyter, html, solutions) (15/01/2018)
5	2	Searching	JH	
6	1	Strings and string search	GG	
6	2	Genetic algorithms	MS	
7	1	No Lecture	–	
7	2	Overall Q/A	GG	

Topic을 누르면 나오는 Course 설명 페이지

Basic Data Structures

Collections

- Like all containers, collections can grow, shrink, or change over time.
- Collections support basic operations such as: *insert* elements to a collection, *remove* elements from a collection, or *search* whether an element belongs to a collection.

```
# Add an element pointed to by x.
insert(C, x)
```

```
# Given a pointer x to an element in a collection C, remove x from C.
remove(C, x)
```

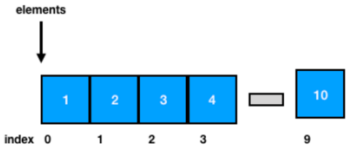
```
# Given a collection C and a key k, return a pointer x to an element in C.
# Output: x.key = k or NIL
search(C, k)
```

Arrays

An array is a container that can hold a *fixed* number of items that should be of the same type.

```
arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

- Each array item is called an *element*.
- The location of each element, in an array, can be identified using numerical *indices*.
- A basic property of an array is its *size*



Assignment 파일을 클릭하여 다운받은 후 실행 화면