Абстрактний тип даних – такий тип даних, який приховує свою внутрішню реалізацію від клієнта. При розробці абстрактного типу даних слід завчасно подумати про інтерфейс, адже він не повинен залежати від представлення даних всередині АТД. Після визначення операцій, що будуть визначати інтерфейс, потрібно зосередитись на даних, які будуть реалізувати потрібну поведінку АТД. Це дозволить нам отримати структуру потрібних даних, тобто механізм, який дозволить зберігати і обробляти дані. При цьому одна з головний переваг використання АТД у цій курсовій роботі – це те, що в разі, коли нам буде потрібно змінити внутрішнє представлення даних то не потрібно буде змінювати всю програму щоб підлаштувати її під цей тип даних, натомість АТД дозволяє міняти роботу об’єктів цього типу, а не всієї програми.

А взагалі, основні переваги використання АТД можна описати наступним чином:

* Інкапсуляція

Після реалізації проекту, ми надаємо користувачу інтерфейс програми, за допомогою якого йому буде зручно взаємодіяти з програмою. Водночас, змінивши певні деталі інтерфейсу, користувач не змінить базової начинки програми і це сприятиме відсутності помилок, збоїв через некваліфіковане використання.

* Інформативність інтерфейсу

АТД дозволяє представити інтерфейс на заданій предметній області.

* Зменшення складності

Завдяки абстрагування від деталей реалізації, ми можемо зосередитись на тому, що може робити АТД, а не на тому, як це робиться.

* Обмеження області використання даних

При використанні АТД ми можемо бути впевнені, що внутрішні дані не будуть залежати від інших ділянок коду, що забезпечить незалежність АТД

Про АТД слід думати перед створенням класу хоча б для того, щоб спочатку визначити і скласти інтерфейс майбутнього класу. Для визначення того, які дані потрібно приховати, а яким надати відкритий доступ. Потрібно серйозно задуматись про забезпечення інтерфейсу високою інформативністю і легкістю оптимізації.

Оскільки, у своїй курсовій роботі я буду в основному працювати з даними типу JSON, то з цього можна визначити АТД, з якими мені в основному доведеться стикатись. Це: list, dictionary, string and integer.

class Stack:

def \_\_init\_\_(self):

self.items = []

def IsEmpty(self):

return self.items == []

def push(self,item):

self.items.append(item)

def pop(self):

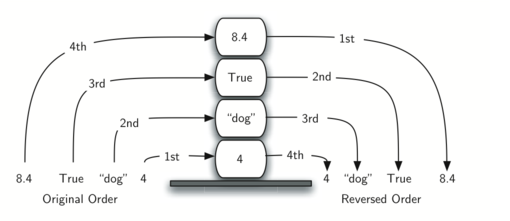
return self.items.pop()

def top(self):

return self.items[len(self.items)-1]

def size(self):

return len(self.items)



Вище наведені приклад для роботи з АТД stack принципом його роботи. Гадаю, що стек буде корисним у обробці даних для мого проекту. Одна з головних переваг стеку – це принцип зберігання і видалення інформації у ньому. Слід сказати, що ці процеси відбуваються у зворотному порядку: елементи, які додаються першими – видаляються останніми і навпаки: елементи, додані останніми, можуть бути використані першими. І при створенні програми варто звертати увагу на цю особливцість задля забезпечення якнайбільшої швидкодії.