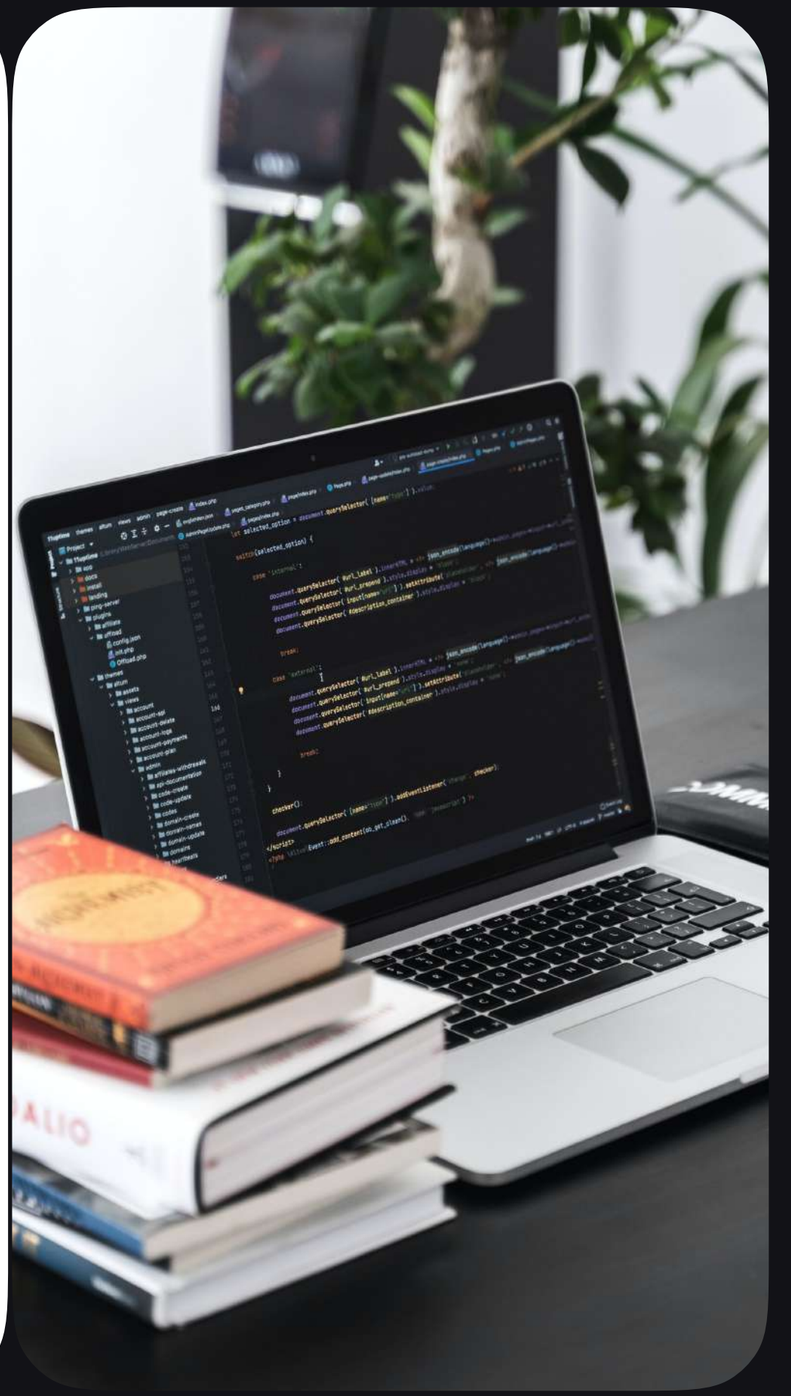
Модуль 2 Занятие 10

Структуры данных в Python



Структуры данных

Структуры данных — это способ организации и хранения информации в памяти компьютера, который позволяет выполнять с ней определенные операции, такие как изменение данных и поиск.

Основные структуры данных: статический массив, динамический массив, связные списки, стек, дек, очередь, хэш-таблицы, дерево, граф.

Как устроена память компьютера

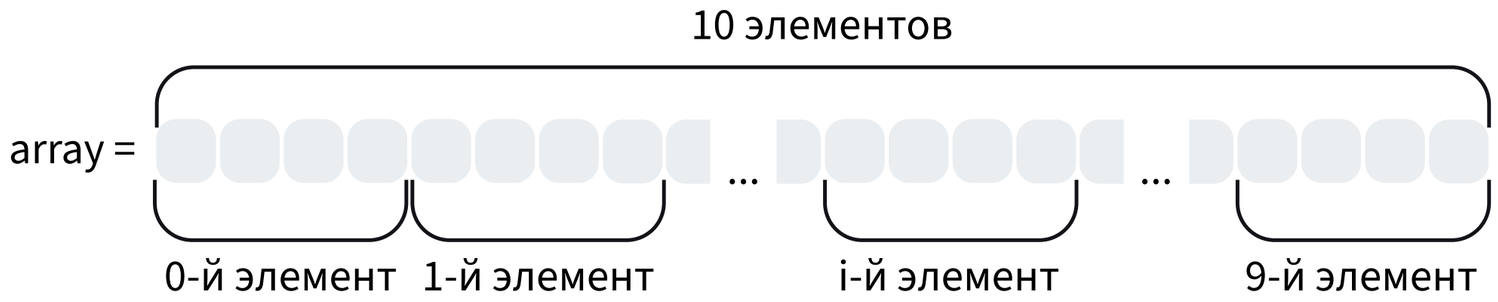
Память компьютера представляет из себя множество ячеек размером в 1 байт. У каждой ячейки есть свой номер — адрес. Процессор использует адреса, чтобы обращаться к данным. Это похоже на то, как наши программы обращаются к значениям переменных по их имени.

Размер ячейки — 1 байт = 8 бит, 8 бит позволяют закодировать $2^8 = 256$ различных вариантов значений.
То есть в одну ячейку памяти можно записать, например, целое число от 0 до 255.

Массивы

Массив (статический массив) — это структура данных, имеющая фиксированный размер и позволяющая хранить значения одного и того же типа. Элементы массива располагаются в памяти друг за другом и занимают один непрерывный участок памяти.

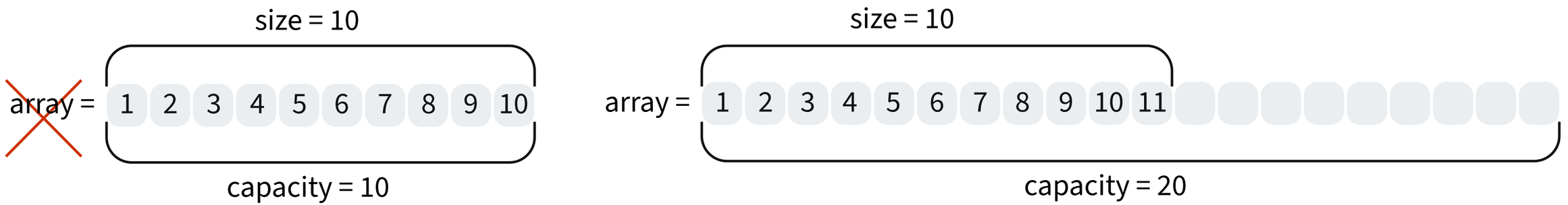
Например, массив array из 10 целых чисел (пусть каждый элемент занимает 4 байта) выглядит так. Имя массива указывает на адрес нулевого элемента. Чтобы узнать адрес следующего элемента, можно прибавить к адресу начала массива размер элемента в байтах. Таким образом можно узнать адрес любого элемента массива.



Динамические массивы

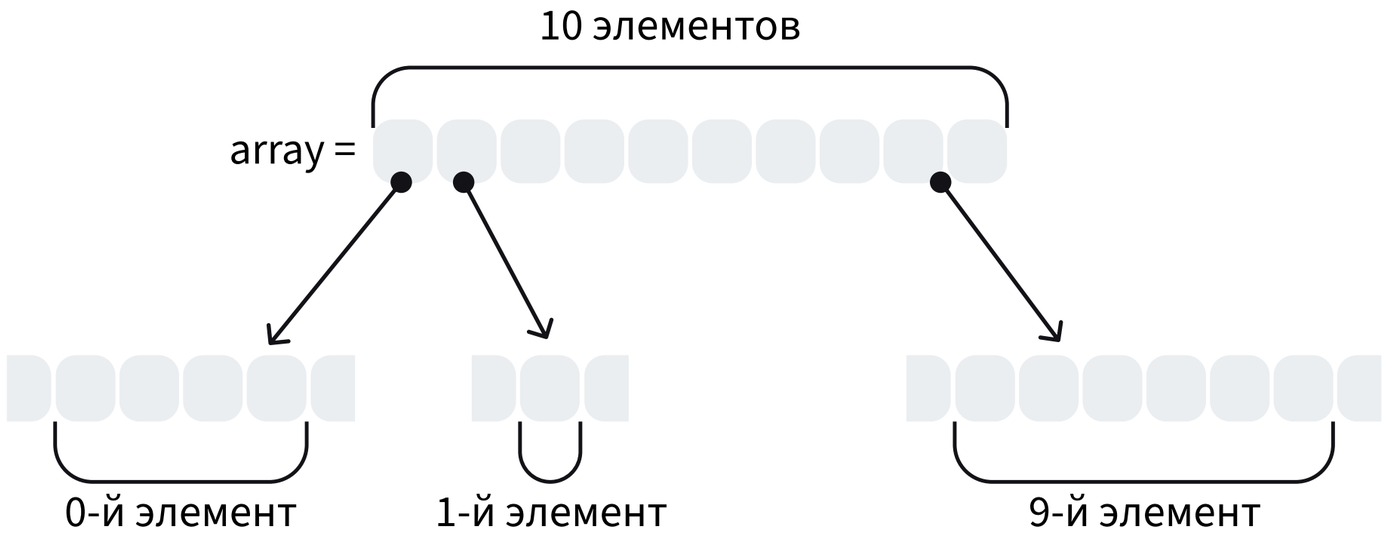
Динамический массив — это массив, который может менять свой размер в процессе работы программы.

Такой массив имеет размер (size) — это количество элементов в массиве и емкость (capacity). Если при добавлении нового элемента size становится больше capacity, то создается новый массив с большей емкостью (чаще всего capacity * 2) и в него копируются все элементы исходного массива. Такая операция называется релокация — выделение нового участка памяти и копирование туда содержимого исходного массива.



Динамические массивы в Python

В Python для создания динамического массива можно использовать конструкцию list (список). Но ведь в списке можно хранить значения разных типов? Дело в том, что в списке хранятся не сами значения, а только ссылки на них — адреса, а адрес всегда занимает один и тот же размер, на какой бы объект он не указывал.



Динамические массивы в Python

```
array = []
array.append(1)
array.append(2)
array.append(3)
array.insert(1, 'new')
print(array)
print(array.index(3))
array.pop(0)
del array[0]
print(array)
```

Вывод:

```
[1, 'new', 2, 3]
3
[2, 3]
```

Для добавления элемента в конец используется метод append(). Для вставки в произвольную позицию метод insert(). Для поиска метод count(). Для удаления метод pop() или инструкция del.

Преимущества и недостатки

Преимущества:

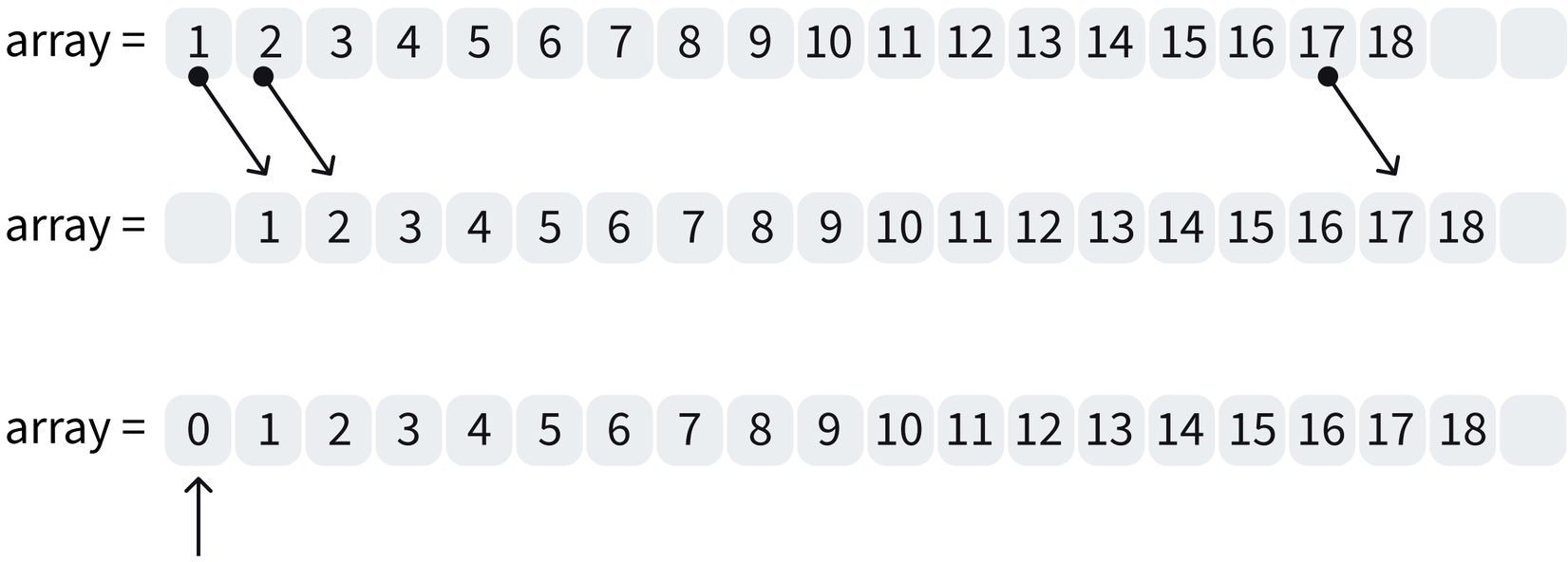
- Быстрый доступ к произвольному элементу
- Простота реализации

Недостатки:

- Операции вставки и удаления в массив требуют больше времени
- Сложность хранения больших массивов за счет использования непрерывного участка памяти

Операция вставки элемента в массив

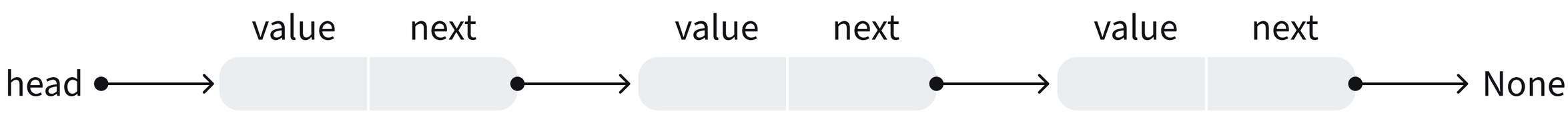
Например, при вставке элемента в начало массива необходимо сдвинуть все существующие элементы на одну позицию вправо, а затем вставить элемент в начало массива.



Связные списки

Существует структура данных, в которой при вставке или удалении элемента передвигать остальные не нужно. Такая структура данных существует — **связный список**.

В связном списке каждый элемент имеет значение и ссылку на следующий элемент списка. Последний элемент ссылается в никуда (в Python можно использовать None). Ссылку на начало связного списка называют головой списка (head).



Связные списки в Python

```
class Node:
    def __init__(self, value, next):
        self.value = value
        self.next = next


n3 = Node('third', None)
n2 = Node('second', n3)
n1 = Node('first', n2)
```

Каждый элемент связного списка представляет из себя экземпляр класса Node (узел), с двумя атрибутами: value (значение) и next (следующий элемент).

Для вывода связного списка воспользуемся функцией print_linked_list:

```
def print_linked_list(vertex):
    while vertex:
        print(vertex, end=" -> ")
        vertex = vertex.next
    print("None")


head = n1
print_linked_list(head)
```

Вывод:

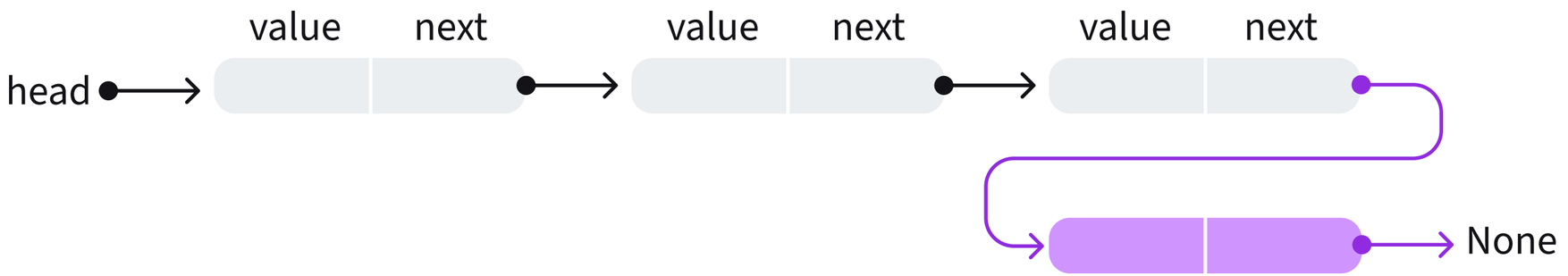
```
first -> second -> third -> None
```

Вставка значения в связный список

Возможно три варианта вставки значения в связный список: в начало, в конец, в произвольную позицию.

Вставка в начало

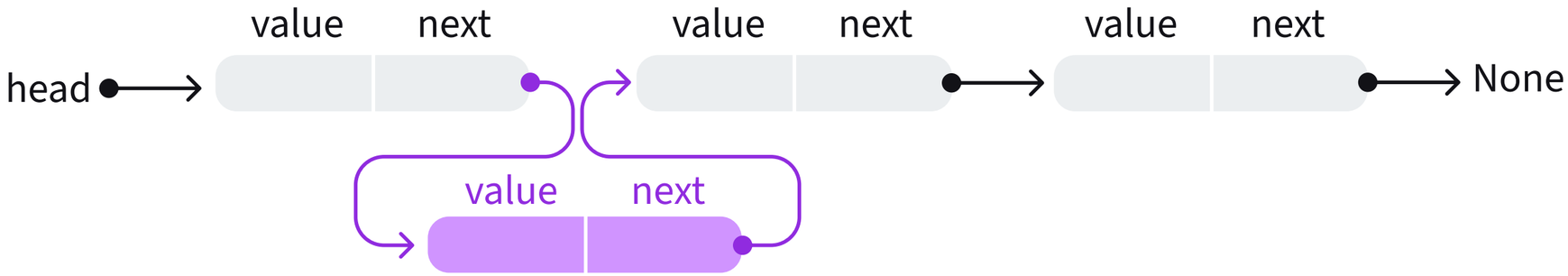
Вставка в конец



Вставка значения в связный список

Вставка в произвольную позицию чуть сложнее. Сначала необходимо найти элемент, за которым необходимо вставить новый, назовём его previous. Затем у нового элемента ставим next на тот элемент, на который указывал previous.next, а previous.next изменяем на новый, только что вставленный элемент.

Вставка в произвольную позицию



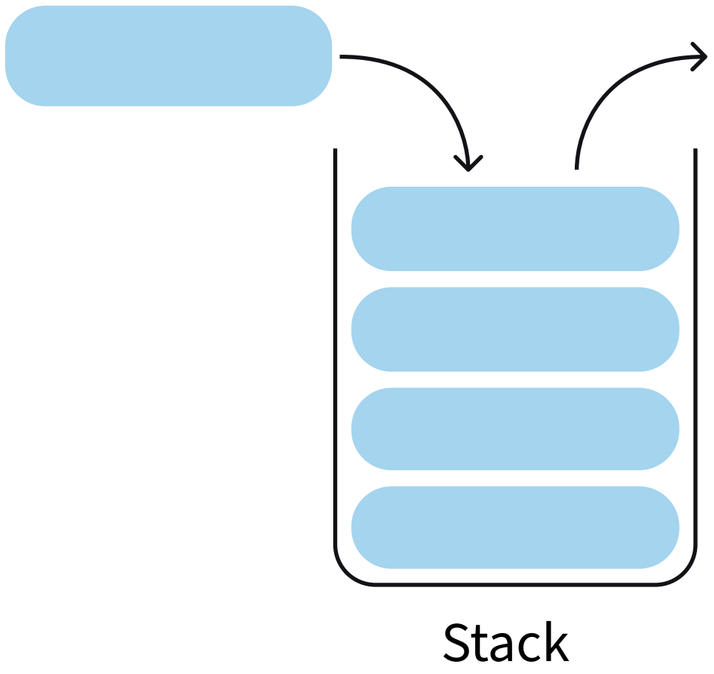
Вставка значения в связный список в Python

Рассмотрим программу вставки значения в связный список

Открыть код

Стек

Стек (Stack) — структура данных, которая позволяет быстро добавить элемент в конец и быстро получить последний элемент. Стек очень часто применяется в программировании и в областях связанных с компьютерами. Стек работает по принципу LIFO (Last In First Out, последним вошел — первым вышел).



Стек

Стек — это удобный интерфейс взаимодействия с данными, а не особенность их хранения. Стек можно реализовать на массиве, а можно на связных списках. Для стека обычно определяют следующие методы:

- push(item) — добавляет элемент в стек (на вершину)
- pop() — возвращает элемент со стека (с вершины) и удаляет его
- size() — возвращает размер стека (количество элементов

Реализация стека в Python

```
class Stack:
    def __init__(self):
        self.items = []

    def push(self, item):
        self.items.append(item)

    def pop(self):
        return self.items.pop()

    def size(self):
        return len(self.items)

    def __str__(self):
        return f'{self.items}'

stack = Stack()
stack.push(1)
stack.push(2)
stack.push(3)
print(stack)
```

Создадим класс Stack

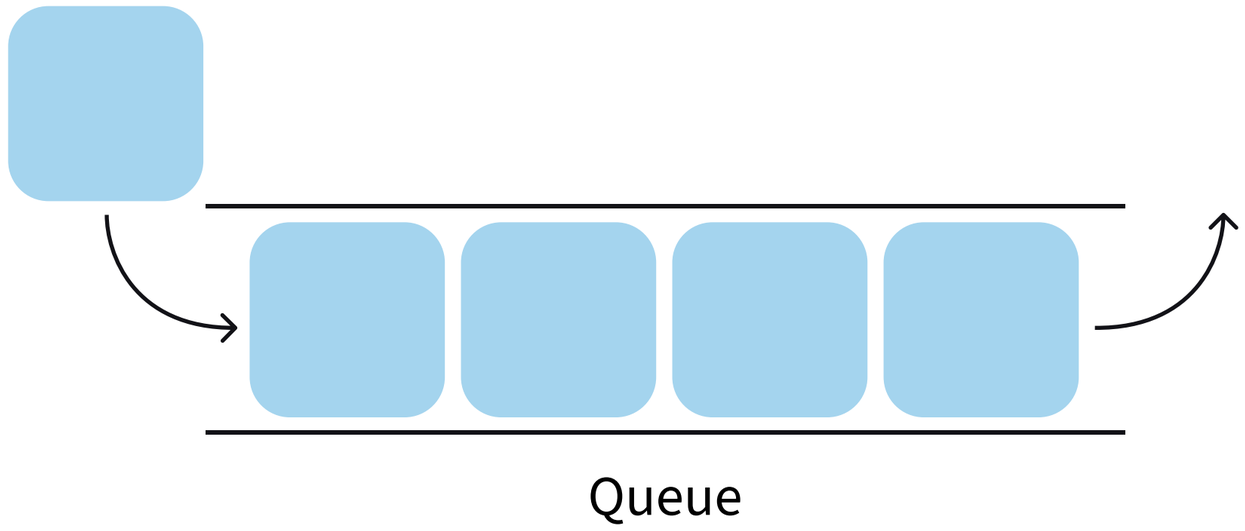
Для хранения элементов будем использовать список. Добавим необходимые методы для работы со стеком: push(), pop(), size().

Вывод:

```
[1, 2, 3]
```

Реализация стека в Python

Очередь (Queue) — структура данных, которая позволяет быстро добавить элемент в конец и быстро получить первый элемент. Очередь работает по принципу FIFO (First In First Out, первым вошел — первым вышел).



Queue

Очередь

Очередь, как и стек, предоставляет нам интерфейс взаимодействия с данными и не накладывает ограничений на способ хранения этих данных.

Для очереди обычно определены следующие методы:

- push(item) — добавляет элемент в очередь (в конец)
- pop() — возвращает элемент из очереди (с начала) и удаляет его
- size() — возвращает размер очереди (количество элементов)

Дек

Дек (Deque) — double ended queue или очередь с двумя концами. Структура данных предоставляющая интерфейс, который позволяет и добавлять, и извлекать элементы с обоих концов.

Для очереди обычно определены следующие методы:

- push_back(item) — добавляет элемент в конец дека
- pop_back() — возвращает последний элемента дека и удаляет его
- push_front(item) — добавляет элемент в начало дека
- pop_front() – возвращает первый элемента дека и удаляет его
- size() — возвращает размер дека (количество элементов)

Итоги

- Структуры данных — это способ организации и хранения информации в памяти компьютера, который позволяет выполнять с ней определенные операции, такие как изменение данных и поиск.
- Динамический массив — это массив, который может менять свой размер в процессе работы программы. В Python для создания динамического массива можно использовать конструкцию list (список).
- Связный список — это структура данных, в которой каждый элемент имеет значение и ссылку на следующий элемент связного списка.
- Стек, очередь, дек — это структуры данных, которые позволяют быстро добавлять элементы в конец или в начало и быстро получить последний или первый элемент. Эти структуры представляют нам интерфейс взаимодействия с данными и не накладывают ограничений на способ хранения этих данных.