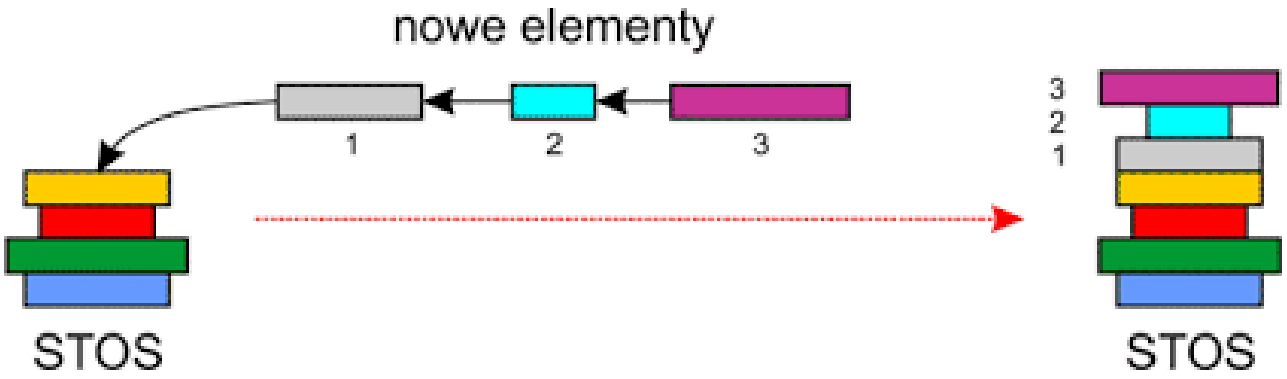
1 Stos - teoria

Stos to liniowa struktura danych. Najprościej możemy go sobie wyobrazić jako stos książek na biurku. W danej chwili możemy wykonać jedną z dwóch operacji:

- push(item) - dodanie elementu na wierzchołek stosu (w tym przypadku książki), wtedy stos rośnie w górę
- pop() - zdjęcie elementu z wierzchołka stosu (elementu znajdującego się na samej górze), wtedy stos maleje



https://eduinf.waw.pl/inf/alg/001_search/images/0100_01.gif

Grafika obrazująca kolejność dodawania kolejnych elementów na stos.

https://eduinf.waw.pl/inf/alg/001_search/images/0100_02.gif

Grafika obrazująca kolejność zdejmowania kolejnych elementów ze stosu.

Warto zwrócić uwagę na fakt, że książki zawsze zdejmujemy ze stosu w kolejności odwrotnej do ich umieszczania – jako pierwszą zdejmujemy książkę, która znalazła się na stosie jako ostatnia. Ze względu na ten fakt stos nazywany jest także kolejką **lifo** - *last in first ou*t.

Godny uwagi jest również fakt, że w tej strukturze danych nie mamy bezpośredniego dostępu do elementu, który nie jest na szczycie. Aby dostać się do innego elementu niż wierzchołek, należy zdjąć wszystkie, które są nad nim.

Stos jest używany w systemach komputerowych na wszystkich poziomach funkcjonowania systemów informatycznych. Stosowany jest przez procesory do chwilowego zapamiętywania rejestrów procesora, do przechowywania zmiennych lokalnych, a także w programowaniu wysokopoziomowym.

2 Stos - implementacja

W przedstawionej obiektowej implementacji wdrożone zostały następujące metody:

- **push**(item) - dodanie elementu na stos
- **pop**() - zdjęcie elementu ze stosu (usuwa element ze stosu oraz zwraca jego wartość)
- **isEmpty**() - sprawdzenie, czy stos jest pusty
- **size**() - zwrócenie liczby elementów znajdujących się na stosie
- **peek**() - zwrócenie wartości elementu, który został dodany na stos jako ostatni

```
# klasa Stack jest obiektową implementacją stosu
class Stack():
    def __init__(self): # konstruktor tworzący listę, na której oparte jest działanie całej klasy
        self.stack = list()
    def isEmpty(self): # metoda sprawdzająca czy stos jest pusty
        return len(self.stack) == 0
    def push(self,item): # metoda dodająca element na wierzchołek stosu
        self.stack.append(item)
    def pop(self): # metoda zdejmująca ze stosu element, który znajduje się na samej górze stosu
        if self.isEmpty():
            return None
        else:
            return self.stack.pop(-1)
    def peek(self): # metoda zwracajaca wartość elementu, który znajaduje się na samej górze stosu
        if self.isEmpty():
            return None
        else:
            return self.stack[-1]
    def size(self): # metoda zwracająca ilość elementów znajdujących się aktualnie na stosie
        return len(self.stack)
```

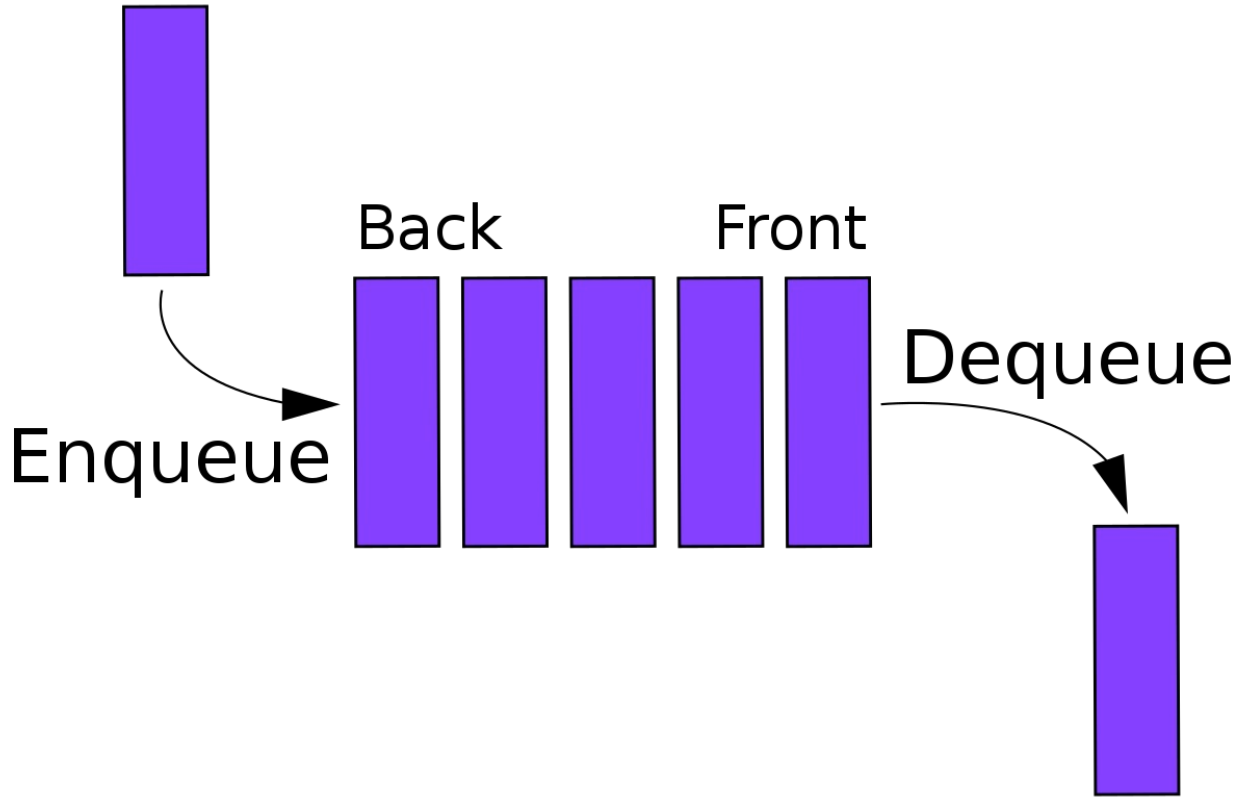
```
my_stack = Stack() # tworzymy obiekt klasy Stack
my_stack.push("Pikachu") # dodajemy na stos kolejne elementy
my_stack.push("Geralt")
my_stack.push("Charmander")
print(my_stack.size()) # out: 3
print(my_stack.pop()) # out: Charmander
print(my_stack.peek()) # out: Geralt
```

Na maturze z informatyki bardzo ważnym aspektem jest czas. Właśnie ze względu na jego ograniczone zasoby nie warto pisać specjalnej klasy. Lepiej zaimplementować stos za pomocą tablicy i korzystać z jej metod wbudowanych.

```
my_stack = list() # tworzymy listę, która posłuży nam jako stos
my_stack.append("Pikatchu") # elementy na stos dodajemy za pomocą metody append
my_stack.append("Geralt")
my_stack.append("Charmander")
print(len(my_stack)) # natomiast do sprawdzania, czy stos jest pusty, oraz do zwracania ilości elementów
# znajdujących się na stosie wykorzystujemy funkcję len (out: 3)
print(my_stack.pop(-1)) # elementy ze stosu zdejmujemy za pomocą metodę pop (out: Charmander)
print(my_stack.pop(-1)) # out: Geralt
```

3 Kolejka – teoria

Kolejka to liniowa struktura danych, w której nowe dane dopisywane są na końcu, natomiast dane do dalszego przetwarzania pobiera się z początku. Nowy element zostaje wstawiony na koniec kolejki i przesuwa się sukcesywnie do przodu podczas przetwarzania wcześniej dodanych elementów. Kolejka jest strukturą danych typu fifo – first in, first out -, a więc na początku kolejki znajduje się zawsze element, który został do niej dodany najwcześniej.



https://upload.wikimedia.org/wikipedia/commons/thumb/5/52/Data_Queue.svg/1200px-Data_Queue.svg.png
Grafika obrazująca działanie kolejki – nowe dane zostają dodane na koniec, natomiast do dalszego przetwarzania pobiera się dane, które znajdują się najdłużej w kolejce

Kolejka jako abstrakcyjny typ danych imituje zachowanie kolejek znanych z codziennego życia (np. kolejek w sklepach do kas).

https://img.freepik.com/premium-wektory/ludzie-w-kolejce-mezczyzna-i-kobieta-stojaca-czeka-w-dlugiej-linii-zatloczona-kolejka-w-koncepcji-sklepu-spozywczego_176516-128.jpg?w=2000
Grafika ilustrująca kolejkę w sklepie.

Zwróćmy uwagę, że w przeciwieństwie do stosu, kolejka nie odwraca kolejności elementów w sekwencji.

W szeroko pojętej informatyce kolejki stosowane są bardzo często. Kolejkę spotyka się między innymi w sytuacjach związanych z różnego rodzaju obsługą zdarzeń. W szczególności w systemach operacyjnych ma zastosowanie kolejka priorytetowa, przydzielająca zasoby sprzętowe uruchomionym procesom.

4 Kolejka - implementacja

W przedstawionej obiektowej implementacji wdrożone zostały następujące metody:

- **add**(item) - dodanie elementu do kolejki
- **pop**() - usunięcie pierwszego elementu z kolejki oraz zwrócenie jego wartości
- **isEmpty**() - sprawdzenie, czy kolejka jest pusta
- **size**() - zwrócenie liczby elementów znajdujących się w kolejce
- **first**() - zwrócenie wartości pierwszego elementu w kolejce
- **last**() - zwrócenie wartości ostatniego elementu w kolejce

```
# klasa Queue jest obiektową implementacją kolejki
class Queue():
    def __init__(self): # konstruktor tworzący listę, na której oparte jest działanie całej kolejki
        self.queue = list()
    def isEmpty(self): # metoda sprawdzająca czy kolejka jest pusta
        return len(self.queue) == 0
    def add(self,item): # metoda dodająca element na koniec kolejki
        self.queue.append(item)
    def pop(self): # metoda usuwająca pierwszy element z kolejki oraz zwracająca jego wartość
        if self.isEmpty():
            return None
        else:
            return self.queue.pop(0)
    def first(self): # metoda zwracajaca wartość pierwszego elementu koleji, czyli elementu, który znajaduje się w niej najdłużej
        if self.isEmpty():
            return None
```

```
        else:
            return self.queue[0]
    def last(self): # metoda zwracajaca wartość ostatniego elementu koleji, czyli elementu, który znajaduje się w niej najkrócej
        if self.isEmpty():
            return None
        else:
            return self.queue[-1]
    def size(self): # metoda zwracająca ilość elementów znajdujących się aktualnie w kolejce
        return len(self.queue)


my_queue = Queue() # tworzymy obiekt klasy Queue
my_queue.add("Pikachu") # dodajemy do kolejki kolejne elementy
my_queue.add("Geralt")
my_queue.add("Charmander")
print(my_queue.size()) # out: 3
print(my_queue.pop()) # out: Pikachu
print(my_queue.peek()) # out: Geralt
```

Na maturze z informatyki bardzo ważnym aspektem jest czas. Właśnie ze względu na jego ograniczone zasoby nie warto pisać specjalnej klasy. Lepiej zaimplementować kolejkę za pomocą tablicy i korzystać z jej wbudowanych metod.

```
my_queue = list() # tworzymy listę, która posłuży nam jako kolejka
my_queue.append("Pikatchu") # elementy do kolejki dodajemy za pomocą metody append
my_queue.append("Geralt")
my_queue.append("Charmander")
print(len(my_queue)) # natomiast do sprawdzania, czy kolejka jest pusta oraz do zwracania ilości #elementów  znajdujących się w kolejce wykorzystujemy funkcję len (out: 3)
print(my_queue.pop(0)) # za pomocą metody pop usuwamy z kolejki pierwszy element oraz zwracamy #jego wartość (out: Pikatchu)
print(my_queue.pop(0)) # out: Geralt
```