



Типизация и структуризация данных

Организация данных

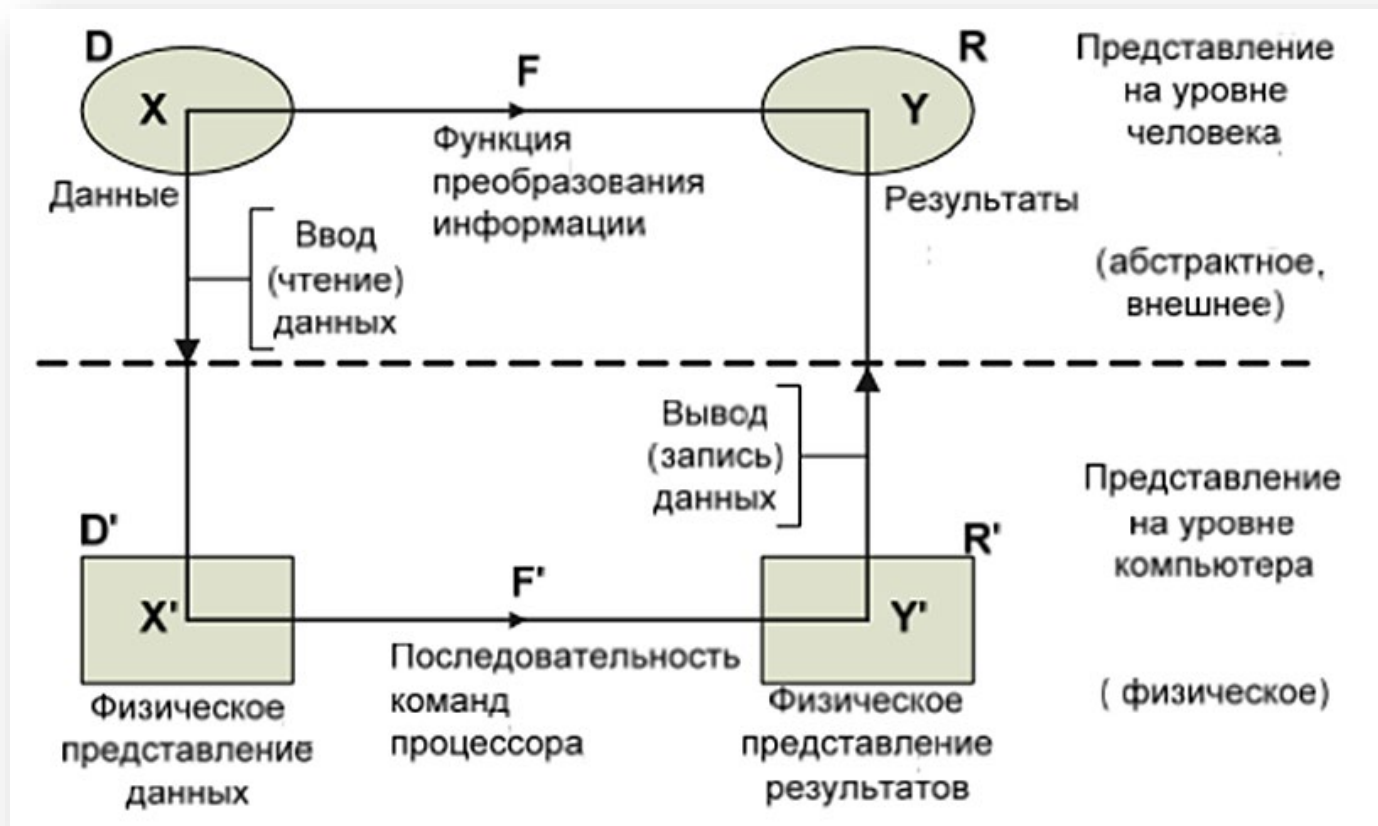
- **Данные** – это представление фактов и идей в формализованном виде, пригодном для передачи и переработке в некоем процессе
- **Информация** - это смысл, который придается данным при их представлении
- **Организация данных** – представление данных и управление данными в соответствии с определенными соглашениями

Проблема

- Есть много данных. Как хранить их внутри приложения?

Правильный ответ зависит от того, как именно вы будете пользоваться этими данными

Модель обработки информации



- Обработка информации – это практическая реализация некоторой функции F , которая отображает множество данных D во множество возможных результатов R .
 - F – произвольная функция, которую надо «вычислить», например, перевод текста с русского на английский, нахождение максимума, расчет траектории ракеты, построение оптимального плана и т.д.

Организация данных

- **Представление данных (Data representation)** – характеристика, выражающая
 - правила кодирования элементов
 - и образования конструкций данных на конкретном уровне рассмотрения в вычислительной системе
- **Управление данными (Data management)** – совокупность функций обеспечения
 - требуемого представления данных
 - накопления и хранения
 - обновления и удаления
 - поиска по заданному критерию и выдачи данных

Организация данных

- Представление данных (Data representation)
- Управление данными (Data management)

Данные в структурах данных



Данные в файлах



Данные в базе данных (БД)

JSON

CSV

Oracle
MySQL
Microsoft SQL Server
PostgreSQL

Организация данных

- **Представление данных** (Data representation) – характеристика, выражающая
 - **Управление данными** (Data management) – совокупность функций обеспечения
- При постановке задачи необходимо выбрать некоторое абстрактное представление предмета рассмотрения, т.е. определить множество данных, отражающих реальную ситуацию (**модель предметной области**)

Пример. Точка на плоскости

Предметная
область – точка

Программная
реализация –
точка?



Как реализовать эти точки в программе?

Уровни организации данных

- Логическая организация данных: проектный уровень
 - отражает взгляд пользователя на данные
 - применяются формальные методы описания динамически изменяющихся структур
- Представление данных: уровень языка реализации
 - описание данных на языке программирования
- Физическая организация данных
 - учитывается размещение и связь данных в среде хранения

Понятие о типизации языка

Тип объекта

- С машинной точки зрения

Форма представления его значений в памяти.

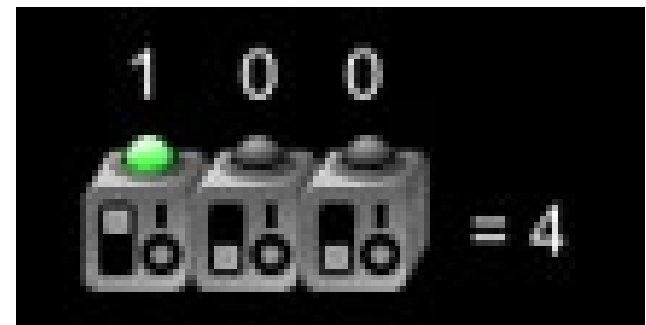
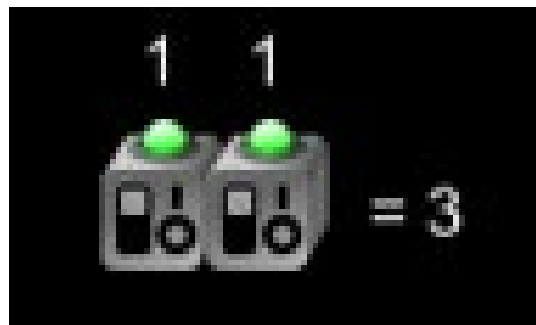
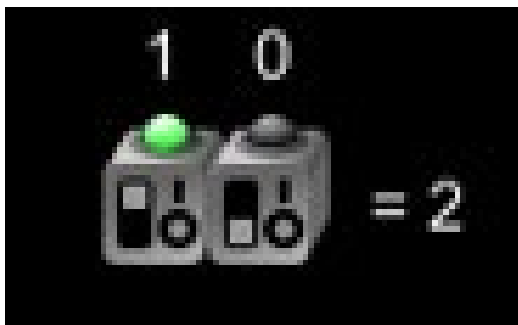
Определяется способ доступа к объекту и его части.

- С точки зрения разработчика

множество значений и набор операций, выполняемых над этими значениями и обладающих некоторыми свойствами

Хранение данных

- Память компьютера может сохранять только два числа: 0 и 1
- Как хранить другие числа?



Двоичная система используется в компьютерах для хранения информации

Биты и байты

- Все биты в памяти поделены на группы
- Каждая группа состоит из 8 бит и называется "байт" (byte)
- Нельзя записать в память меньше, чем один байт



- Как хранится, например число 5?

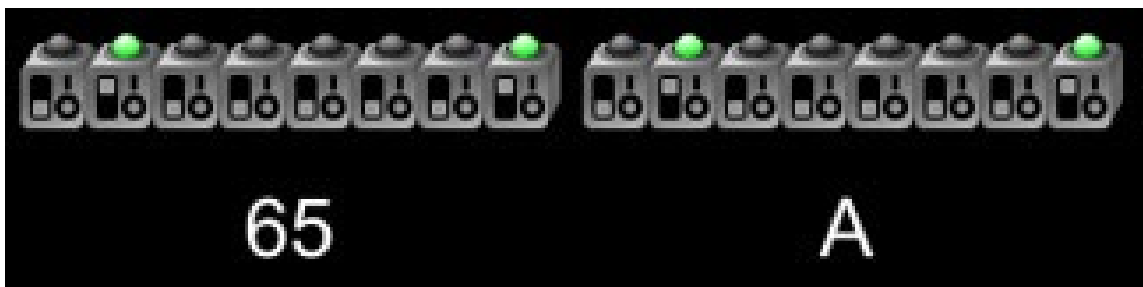
Выравнивание по границе машинного слова (1 байт)



Как компьютер понимает типы данных

- В памяти компьютера хранятся только числа.

- ☐ Никаких букв, никаких красных или синих цветов – только числа
- ☐ Латинская буква "А" обозначается числом 65



- Компьютер не понимает, где что!
- Для него есть только два числа, записанные в два разных адреса. Кто из них число, а кто буква – известно только нам, и вся ответственность лежит на нас

Взаимодействие между типами

- $65 + A = 130$ – хорошо ли это?
- В программировании часто требуется обратный эффект – чтобы мы не могли случайно или нарочно сложить цифру с буквой.
- Языки программирования по-разному реагируют на подобную операцию

Контроль типов

- Основная функция типов
 - обеспечение более полной и легкой проверки правильности программ
- Проверка заключается
 - в определении типов выражений
 - и их согласованности с типами, которые требуются по правилам языка

Такая проверка называется ***контролем типов***

Правила типизации

Программа называется **типово-правильной**, если она удовлетворяет правилам типизации языка:

- Приписывание типов переменным и константам
- Определение типов выражений по типам их частей
- Согласование типов частей языковых конструкций

Язык программирования является типизированным, если для него определены правила типизации

Статическая типизация

- переменная, параметр подпрограммы, возвращаемое значение функции связывается с типом в **момент объявления** и тип не может быть изменён позже
 - Ада, C++, Паскаль

Динамическая типизация

- переменная связывается с типом в **момент присваивания значения**, а не в момент объявления переменной
 - Python, Ruby, PHP, Perl, JavaScript

Уровни типизации

- **Слабо типизированный** (нестрогая типизация) – если информация и типе используется только для обеспечения корректности программы на машинном уровне (ПЛ/1, Алгол-68, Си и С++)
 - разрешается выполнение некорректных операций
 - повышает гибкость языка, но уменьшает понятность и надежность программ.
- **Сильно типизированный** (строгая типизация) – если осуществляется полный контроль типов (язык Ада, С#)
 - повышает надежность и ясность программ

Преимущества типизации

- Модель предметной области лучше структурирована, существует иерархия сортов элементов
- Манипулирование элементами более целенаправленно, разнородные элементы обрабатываются различным образом, однородные – единообразно
- В случае строгой типизации несоответствия типов фиксируются до выполнения программы, гарантируя отсутствие смысловых ошибок и безопасность кода

Тип данных

Определяет

- Формат представления в памяти компьютера
- Множество допустимых значений, которые может принимать принадлежащая к выбранному типу переменная или константа
- Множество допустимых операций, применимых к этому типу.

Простые и структурные типы данных

■ Простые (примитивные)

- ☐ Целочисленные
- ☐ Вещественные
- ☐ Логический тип
- ☐ Символьный тип

■ Структурированные

- ☐ Строка
- ☐ Массив
- ☐ Структура
- ☐ Перечисление
- ☐ Класс

Типы данных C++

| Название | Обозначение | Диапазон значений |
|--|---------------------------------------|---|
| Байт | char | от -128 до +127 |
| без знака | unsigned char | от 0 до 255 |
| Короткое целое число | short | от -32768 до +32767 |
| Короткое целое число без знака | unsigned short | от 0 до 65535 |
| Целое число | int | от - 2147483648 до + 2147483647 |
| Целое число без знака | unsigned int (или просто unsigned) | от 0 до 4294967295 |
| Длинное целое число | long | от - 2147483648 до + 2147483647 |
| Длинное целое число без знака | unsigned long | от 0 до 4294967295 |
| Вещественное число одинарной точности | float | от $\pm 3.4e-38$ до $\pm 3.4e+38$ (7 значащих цифр) |
| Вещественное число двойной точности | double | от $\pm 1.7e-308$ до $\pm 1.7e+308$ (15 значащих цифр) |
| Вещественное число увеличенной точности | long double | от $\pm 1.2e-4932$ до $\pm 1.2e+4932$ |
| Логическое значение | bool | значения true(истина) или false (ложь) |

| Тип C# | Размер в байтах | Тип .NET | Описание |
|--------------------------|-----------------|----------|--|
| Базовый тип | | | |
| object | | Object | Может хранить все что угодно, т.к. является всеобщим предком |
| Логический тип | | | |
| bool | 1 | Boolean | true или false |
| Целые типы | | | |
| sbyte | 1 | SByte | Целое со знаком (от -128 до 127) |
| byte | 1 | Byte | Целое без знака (от 0 до 255) |
| short | 2 | Int16 | Целое со знака (от -32768 до 32767) |
| ushort | 2 | UInt16 | Целое без знака (от 0 до 65535) |
| int | 4 | Int32 | Целое со знаком (от -2147483648 до 2147483647) |
| uint | 4 | UInt | Целое число без знака (от 0 до 4 294 967 295) |
| long | 8 | Int64 | Целое со знаком (от -9223372036854775808 до 9223372036854775807) |
| ulong | 8 | UInt64 | Целое без знака (от 0 до 0xffffffffffffff) |
| Вещественные типы | | | |
| float | 4 | Single | Число с плавающей точкой двойной точности. Содержит значения приблизительно от $\pm 1.5 \cdot 10^{-45}$ до $\pm 3.4 \cdot 10^{38}$ с 7 значащими цифрами |
| double | 8 | Double | Число с плавающей точкой двойной точности. Содержит значения приблизительно от $\pm 5.0 \cdot 10^{-324}$ до $\pm 1.7 \cdot 10^{308}$ с 15-16 значащими цифрами |
| Символьный тип | | | |
| char | 2 | Char | Символы Unicode |
| Строковый тип | | | |
| string | | String | Строка из Unicode-символов |
| Финансовый тип | | | |
| decimal | 12 | Decimal | Число до 28 знаков с фиксированным положением десятичной точки. Обычно используется в финансовых расчетах. |

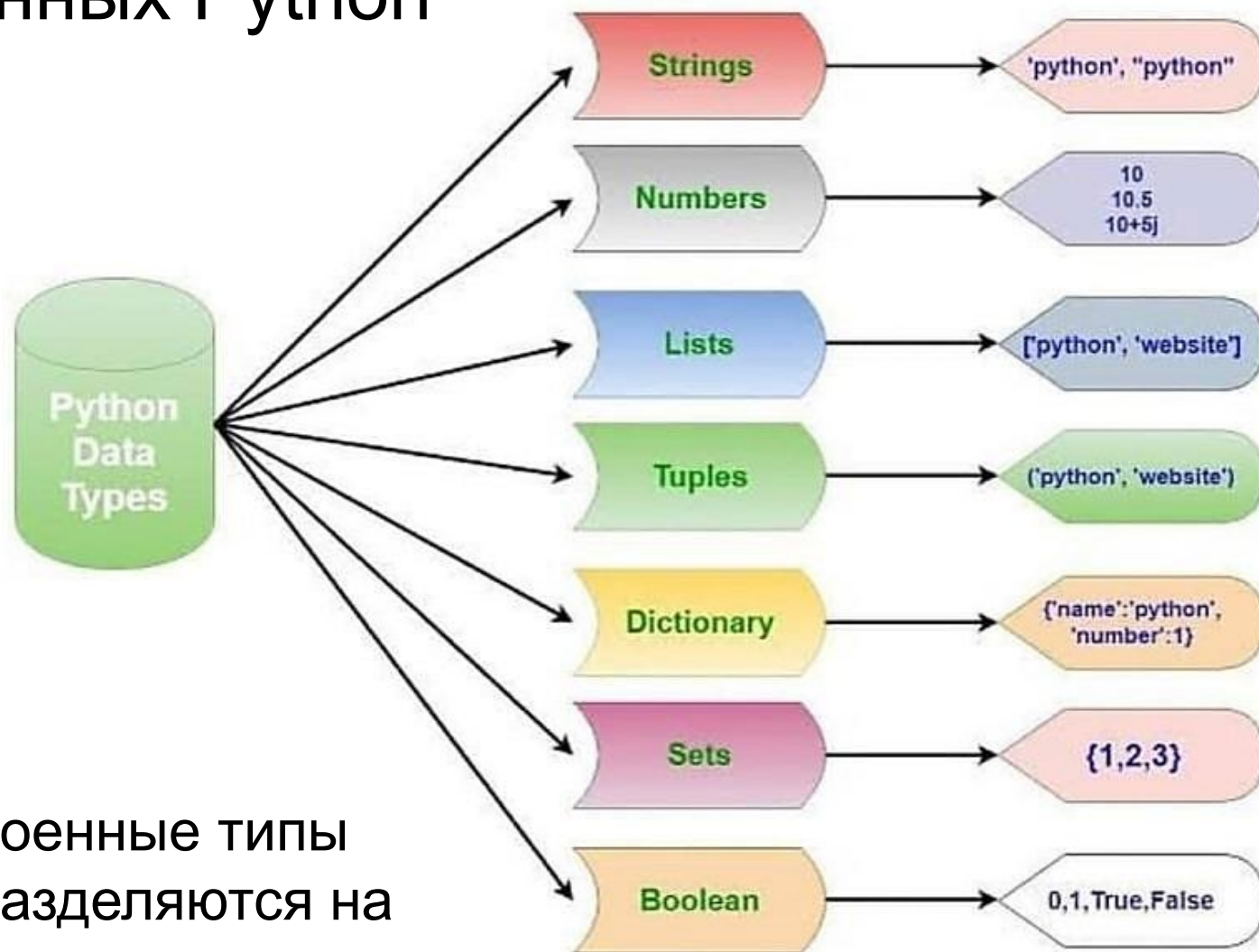
Типы данных Visual Basic

| Тип данных | Размер | Диапазон значений | Префикс |
|-----------------------------|------------------------|--|---------|
| Byte (байт) | 1 байт | От 0 до 255. | byt |
| Boolean (логический) | 2 байт | True или False. | bln |
| Integer (целое) | 2 байт | От -32 768 до 32 767. | int |
| Long (длинное целое) | 4 байт | От -2 147 483 648 до 2 147 483 647. | lng |
| Single | 4 байт | От -3,4E38 до -1,4E-45 для отриц.знач.; от 1,4E-45 до 3,4E38 для пол.знач. | sng |
| Double | 8 байт | От -1,7E308 до -4,9E-324 для отриц.знач.; от 4,9E-324 до 1,7E308 для пол.знач. | dbl |
| Currency (денежный) | 8 байт | С фиксир.точкой От -922 337 203 685 477,5808 до 922 337 203 685 477,5807. | cur |
| Date (даты и время) | 8 байт | От 1 января 100 г. до 31 декабря 9999 г. | dtm |
| Object (объект) | 4 байт | Ссылка на объект (указатель) | obj |
| String (строка перем.длины) | 10 байт + длина строки | От 0 до приблизительно 2 миллиардов. | str |
| String (строка пост.длины) | Длина строки | От 1 до приблизительно 65 400. | str |
| Variant | не менее 16 байт | Любой из перечисл.выше объектов, Null,Error,Empty,Nothing | var |

Примитивные типы данных JavaScript

- **логический** (англ. Boolean) — может принимать два значения- истина (true) и ложь (false);
- **нулевой** (англ. Null) – значение null представляет ссылку, которая указывает, обычно намеренно, на несуществующий или некорректный объект или адрес;
- **неопределённый** (англ. Undefined) – обозначает предопределенную глобальную переменную, инициализированную неопределенным значением;
- **числовой** (англ. Number) – числовой тип данных в формате 64-битного числа двойной точности с плавающей запятой;
- **строковый** (англ. String) – представляет собой последовательность символов, используемых для представления текста;
- **символ** (англ. Symbol) — тип данных, экземпляры которого уникальны и неизменяемы. (новый в ECMAScript 6).

Типы данных Python



В Python встроенные типы данных подразделяются на две группы:

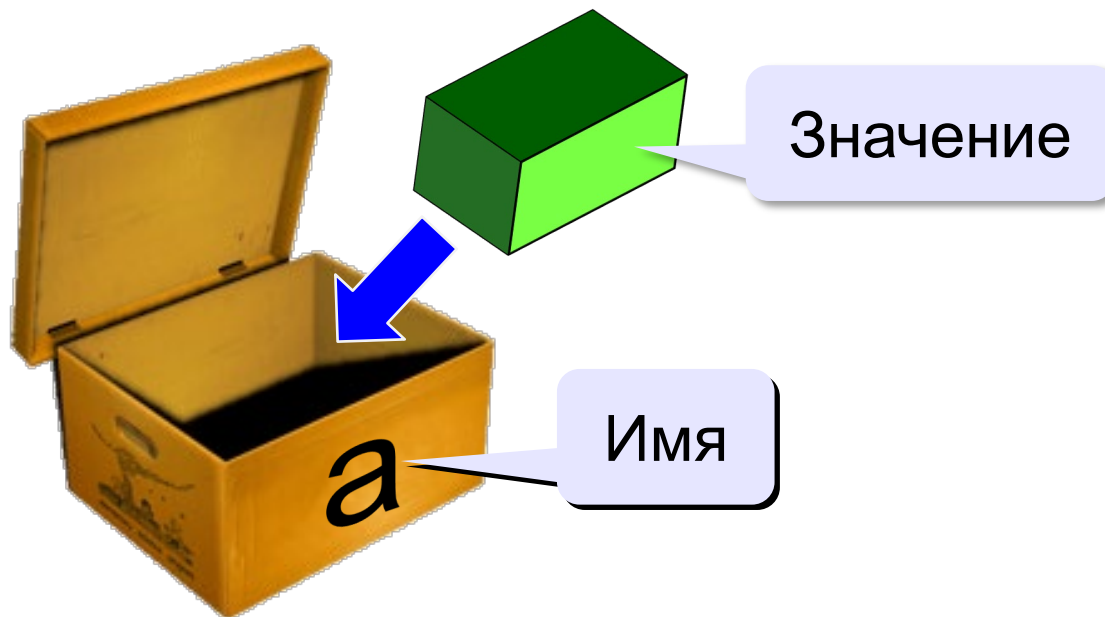
скалярные

структурированные

Переменные

Переменная – это величина, имеющая имя, тип и значение

- Значение переменной можно изменять во время работы программы



Пример

- Известен рост 20 человек. Определить среднее значение роста.
- **Решение.** Для решения можно в программе использовать 20 переменных величин: r_1, r_2, \dots, r_{20} – и, обращаясь к каждой из них по имени, найти сумму значений роста, а затем среднее значение
- **Рекомендация.** Сохранить все введенные значения, а потом их использовать для расчетов



1. Как ввести числа в память?
2. Где хранить введенные числа?
3. Как вычислить?
4. Как вывести результат?

Структурированные типы

Необходимость в структурных типах данных

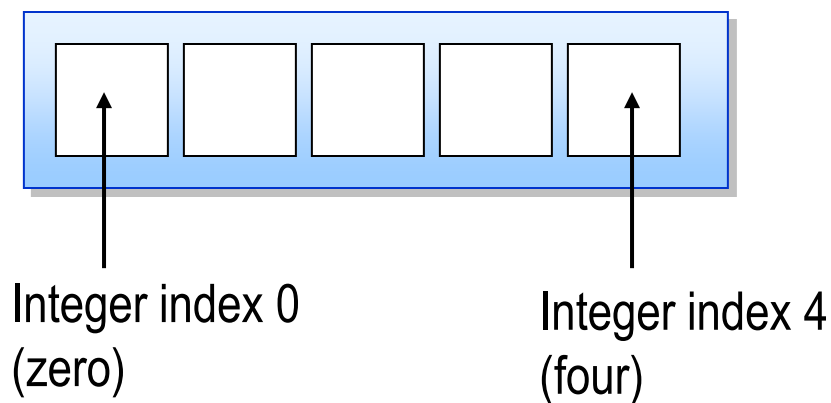
- Для разработки программ методом сверху вниз необходимо иметь возможность описывать данные на различных уровнях
- Данные должны быть структурированы, чтобы их можно было эффективно выбирать

Общее понятие структуры данных

- **Абстрактный тип данных (АТД):**
математическая модель и операции,
определенные в рамках этой модели
- Для представления АТД используются **структуры данных:**
набор переменных, возможно различных типов,
объединенных определенным образом
- **Абстрактные структуры данных** предназначены для удобного хранения и доступа к информации
 - предоставляют удобный интерфейс для типичных операций с хранимыми объектами, скрывая детали реализации от пользователя

Массив

- Массив – последовательный набор элементов
 - Все элементы массива одного типа
 - Доступ к конкретным элементам массива происходит через использование индекса
 - Индексация начинается с нуля



Почему индексация с нуля?

- Первым номером всегда будет 0?
- Первый элемент – всегда первый, и всегда №1, а не №0
- Но его адрес равен нулю (относительно начала)
- Правильно будет так: это первый элемент с адресом 0

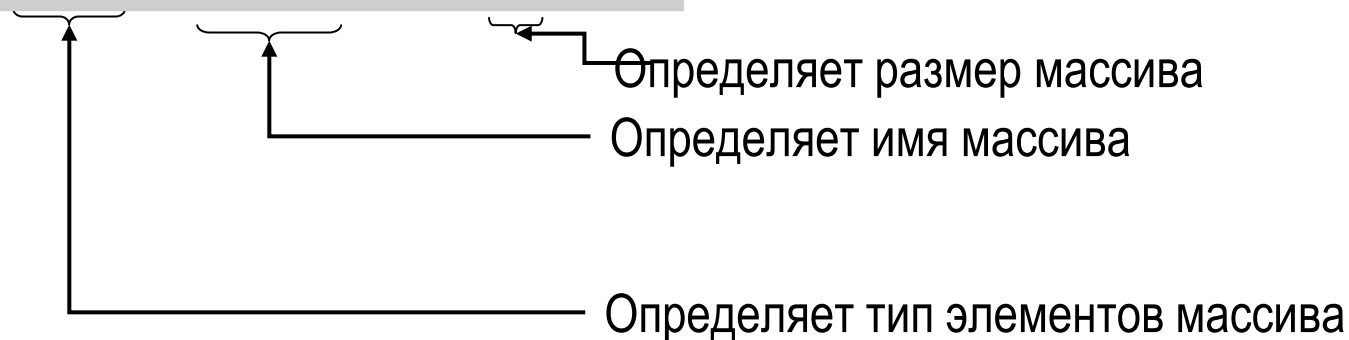


- Адреса – это расстояния от начала памяти
- На каком расстоянии от начала памяти находится первый элемент? На нулевом. Поэтому у него адрес 0

Массив в C++

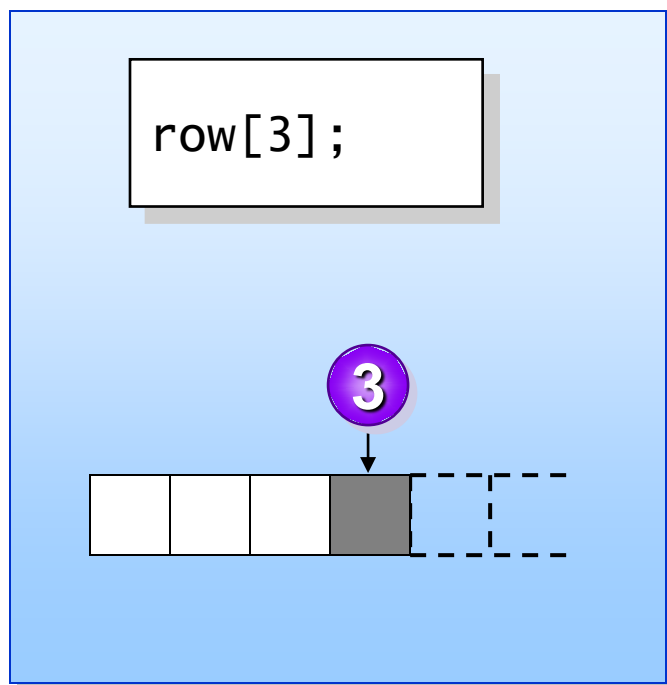
- При объявлении массива необходимо определить:
 - Тип элементов массива
 - Размер массива
 - Имя массива

```
int arrInter[17];
```



Организация доступа к элементам массива

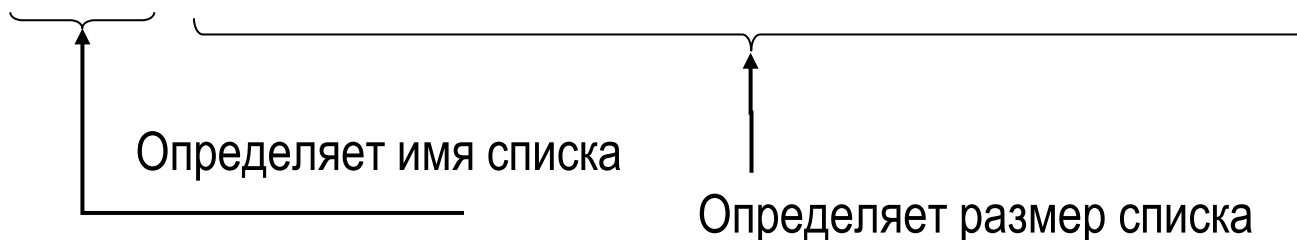
- Определяйте индекс для каждой из размерностей
 - Индекс первого элемента равен нулю



Список в Python

- Список – структура данных, предназначенная для хранения значений (могут быть различного типа):

```
V = [20, 61, 2, 37, 4, 55, 36, 7, 18, 39]
```



- Списки в Python представляют собой непрерывные массивы ссылок на другие объекты
- В CPython списки реализованы в виде массивов переменной длины

| Операция | Сложность |
|--|-----------|
| Копия | $O(n)$ |
| Присоединение | $O(1)$ |
| Вставка | $O(n)$ |
| Извлечение значения элемента | $O(1)$ |
| Установка значения элемента | $O(1)$ |
| Удаление элемента | $O(n)$ |
| Итерация | $O(n)$ |
| Извлечение среза длины k | $O(k)$ |
| Удаление среза | $O(n)$ |
| Установка среза длины k | $O(k+n)$ |
| Расширение | $O(n)$ |
| Умножение на k | $O(nk)$ |
| Проверка существования (элемента в списке) | $O(n)$ |
| <code>min()/max()</code> | $O(n)$ |
| Возврат длины | $O(1)$ |

Организация доступа к элементам списка Python

- Определяйте индекс для каждой из размерностей
 - Индекс первого элемента равен нулю
- Для индексации и получения срезов удобно пользоваться обозначениями:

- `s[-1]` # Последний символ

- Срезы обеспечивают глубокое копирование:

```
x = [53, 68, ["A", "B", "C"]]
```

```
x1 = x # Поверхностная  
копия (через присваивание)
```

```
x2 = x[:] # Глубокая копия (создается при срезе)
```

```
x3 = x.copy() # Глубокая копия (через метод copy())
```

Индексация

| | | | | | | | | | | | | |
|---|----|---|----|---|----|---|----|---|----|---|----|---|
| + | - | + | - | + | - | + | - | + | - | + | - | + |
| | P | | y | | t | | h | | o | | n | |
| + | - | + | - | + | - | + | - | + | - | + | - | + |
| | 0 | | 1 | | 2 | | 3 | | 4 | | 5 | |
| | -6 | | -5 | | -4 | | -3 | | -2 | | -1 | |

Генераторы

- Генераторы списков (списковое включение) – возвращают список

[expression for item in list if conditional]

b = [i+10 **for** i in **a**]

- Генераторы словарей – возвращают словарь:

{ key:value **for** item in list **if** conditional }

- Выражения-генератор (Generator Expressions) – возвращают объект

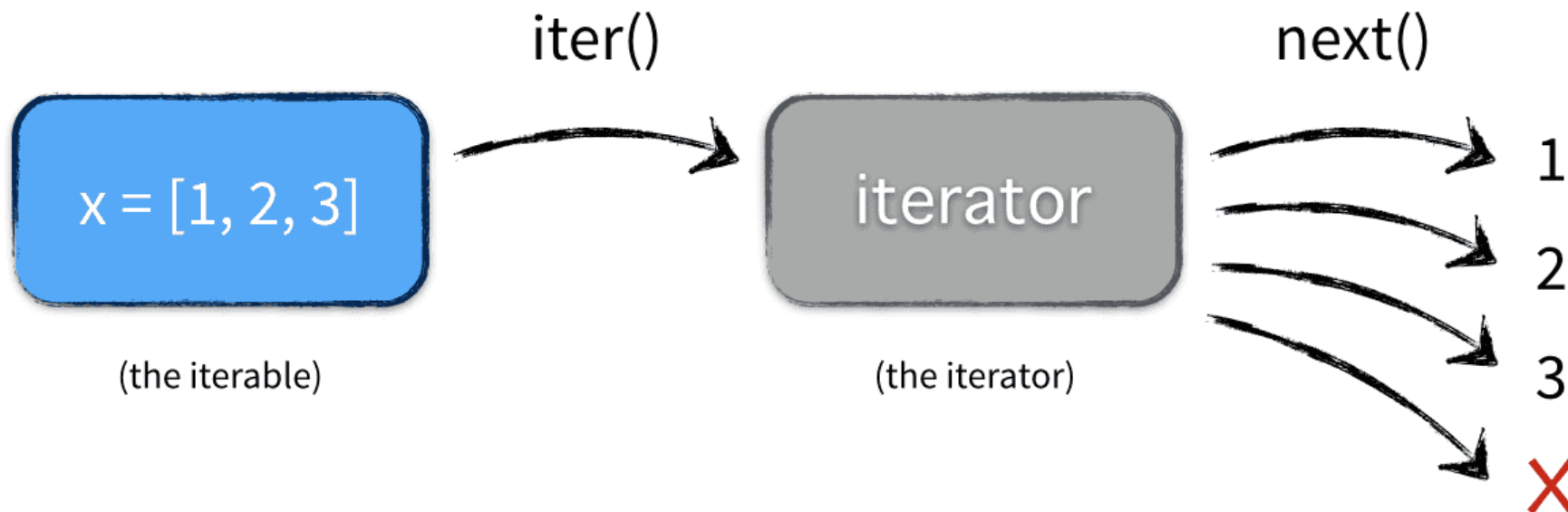
- ☐ генераторы могут быть написаны таким же образом, за исключением того, что они возвращают **объект** генератора, а не список

(expression **for** item in list **if** conditional)

Круглые
скобки

Итераторы

- Итератор - специальный объект, предоставляющий навигацию по другим (итерируемым) объектам



Массивы в Python

- Модуль **array** определяет массивы – с ограничением на тип данных и размер каждого элемента
- Размер и тип элемента определяется при его создании

```
from array import *  
my_array = array('i', [1, 2, 3, 4])
```

signed int, 2 байта

- Массивы изменяемы – поддерживают все списковые методы (индексация, срезы, умножения, итерации)

| Код типа | Тип в C | Тип в python | Минимальный размер в байтах |
|----------|--------------------|--------------|-----------------------------|
| 'b' | signed char | int | 1 |
| 'B' | unsigned char | int | 1 |
| 'h' | signed short | int | 2 |
| 'H' | unsigned short | int | 2 |
| 'i' | signed int | int | 2 |
| 'I' | unsigned int | int | 2 |
| 'l' | signed long | int | 4 |
| 'L' | unsigned long | int | 4 |
| 'q' | signed long long | int | 8 |
| 'Q' | unsigned long long | int | 8 |
| 'f' | float | float | 4 |
| 'd' | double | float | 8 |

Методы массивов (array) в python

- Доступ к отдельным элементам через индексы
- Добавить значение в массив с помощью метода `append()`
- Вставить значение в массив) по любому индексу с помощью метода `insert()`
- Расширение массива с помощью метода `extend()`
- Добавить элементы из списка в массив, используя метод `fromlist()`
- Удалить любой элемент массива, используя метод `remove()`
- Удалить последний элемент массива методом `pop()`
- Получить любой элемент через его индекс с помощью метода `index()`
- Обратный массив, используя метод `reverse()`
- Определить количество вхождений элемента с помощью метода `count()`
- Преобразовать массив в список Python с теми же элементами, используя метод `tolist()`

Перечисления

- Перечисляемый тип представляет собой тип значений, содержащий конечное число именованных констант
- Синтаксис определения перечисления

```
enum <имя> [ : базовый тип]  
{список-перечисления констант(через запятую)};
```

- Создание перечисления

```
enum DayTime { morning, day, evening, night };
```

- Использование перечисления

```
DayTime current;  
if (current != night)  
    // выполнить работу
```

Перечисления в Python

- Модуль **enum** определяет класс перечисления Enum, который может быть использован для уникальных наборов имен и значений (констант)

```
from enum import Enum
class Temperature(Enum):
    minT = 0
    levelT = 55
    maxT = 100
```

Или в функциональном стиле

```
Temperature = Enum('Temperature', {'minT':0, 'levelT':55, 'maxT':100})
```

```
tp = Temperature.levelT.value      # 55
```

```
t1 = 34                                # текущее значение
if t1 < tp:
    print("work")
else:
    print("not work!")
```

Структуры

- Создание структуры

```
public struct Employee  
{  
    string firstName;  
    int age;  
}
```

Структуры могут включать
элементы разных типов

- Использование структуры

```
Employee companyEmployee;  
companyEmployee.firstName = "Joe";  
companyEmployee.age = 23;
```

Структуры

Специального ключевого слова struct (как в C++, C#) в этом языке **нет**

■ Создание структуры в Python

```
# Использование класса для имитации структуры
class Point:
    x = 0
    y = 0
```

```
# создаем точку как переменную-объект класса
point_A = Point()
point_A.x = 1
point_A.y = 5
```

■ Использование словаря

```
# Использование словаря для имитации структуры
# создаем две точки как объекты-словари
pD1 = {'x':1, 'y':5}
pD2 = {'x':4, 'y':7}
```

Динамические структуры данных

Динамические структуры данных – это структуры данных, память под которые выделяется и освобождается по мере необходимости

■ Особенности:

- ☐ отсутствие физической смежности элементов структуры в памяти
- ☐ в процессе существования в памяти могут изменять не только число их элементов и их значений, но и характер связей между элементами

■ Элемент динамической структуры состоит из двух полей

- ☐ информационного поля или поля данных
- ☐ поле связей

Связное представление данных

Достоинства:

- размер структуры ограничивается только доступным объемом машинной памяти
- при изменении логической последовательности элементов структуры требуется не перемещение данных в памяти, а только коррекция указателей
- большая гибкость структуры

Недостатки:

- на поля связей расходуется дополнительная память
- доступ к элементам связной структуры может быть менее эффективным по времени

Реализация структур данных

- В языках программирования имеется возможность явно запрашивать и использовать области **динамической памяти**.
- C++
 - **Указатель** содержит адрес поля в динамической памяти, хранящего величину определенного типа.
 - Сам указатель располагается в статической или автоматической памяти
- C#
 - Коллекция – группа объектов
 - Коллекции упрощают реализацию многих задач программирования, предлагая уже готовые решения для построения структур данных

Коллекции общего назначения

Реализуют структуры данных:

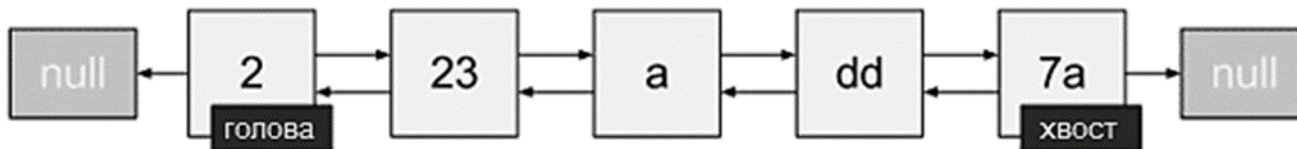
- Стеки
- Очереди
- Динамические массивы
- Словари (хеш-таблицы, предназначенные для хранения пар ключ/значение)
- Отсортированный список для хранения пар ключ/значение

Простейшие структуры данных

- **Список** - упорядоченное множество, состоящее из переменного числа элементов, к которым применимы операции включения, исключения.
- **Линейный список** - список, отражающий отношения соседства между элементами

Array vs. Linked List

Связный список (LinkedList)



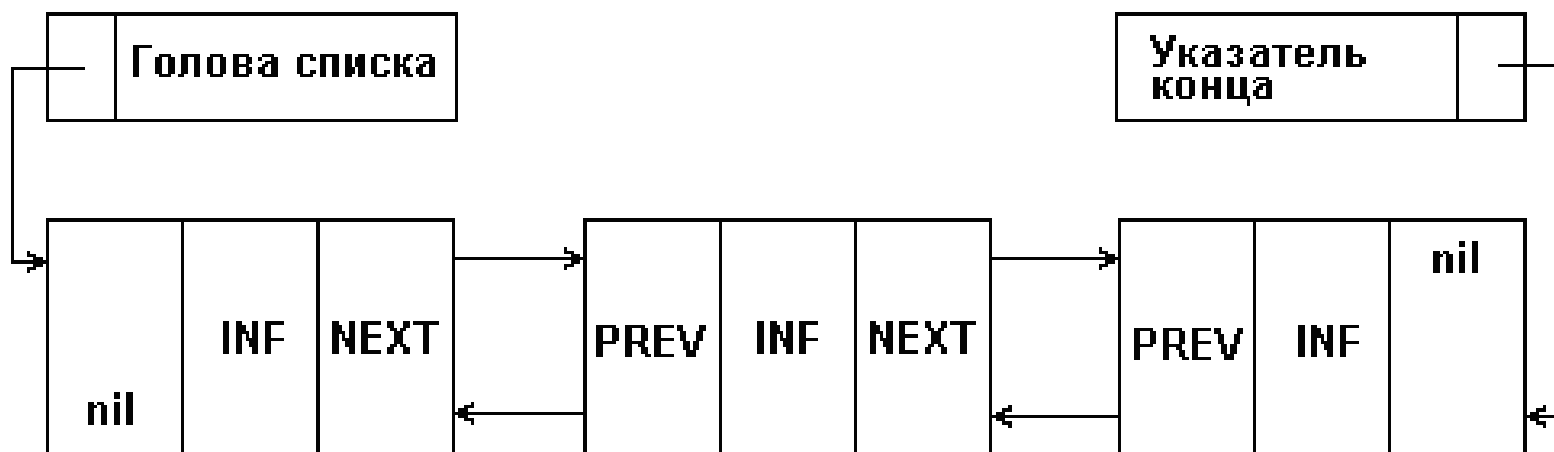
Массив (Array и ArrayList)



Представление односвязного списка в памяти

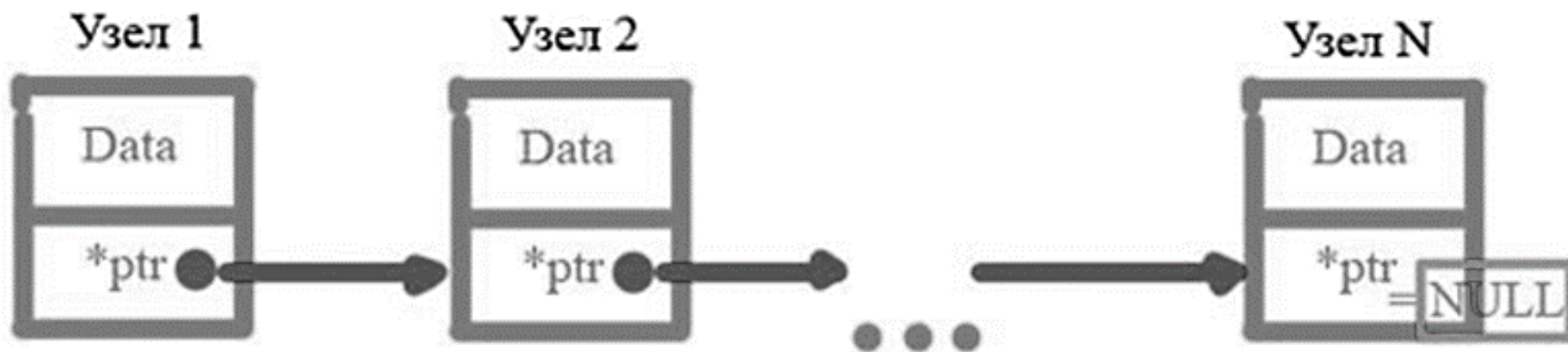


Представление двусвязного списка в памяти



Реализация однонаправленного связанного списка

- Связный список - это набор элементов, содержащихся в узлах, каждый из которых также содержит ссылку на некоторый узел

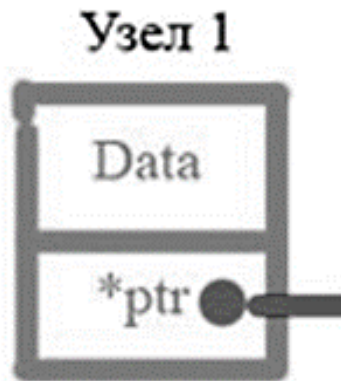


Data - это какие-либо данные в узле

*ptr - это указатель, содержащий адрес следующего узла в памяти компьютера

Реализация однонаправленного связанного списка

- Узел динамического списка – структура из двух полей
- данные и указатель на структуру того же типа



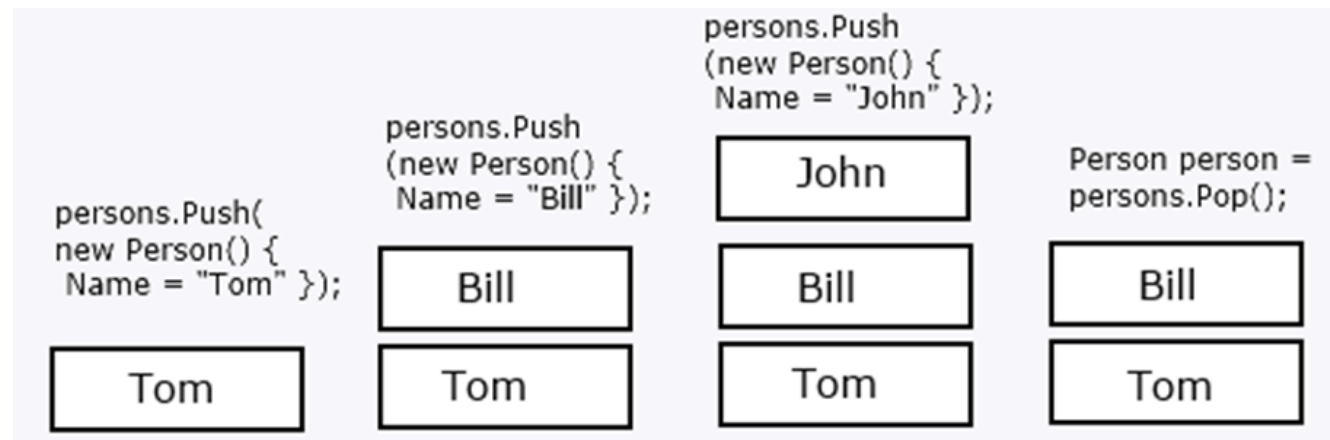
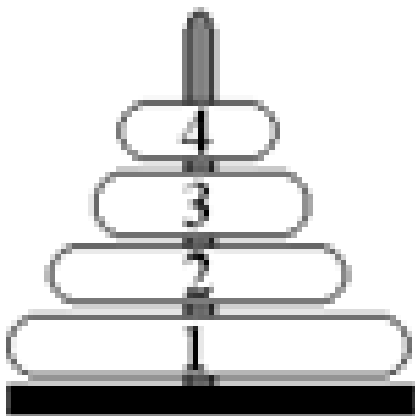
```
struct Node  
{  
    int data;  
    Node *next;  
};
```

Не существует никаких ограничений на тип

Ссылка на следующий элемент списка

Стек

- **Стек** - такой последовательный список с переменной длиной, включение и исключение элементов из которого выполняются только с одной стороны списка, называемого вершиной стека.
- LIFO (Last – In – First - Out - "последним пришел - первым исключается").
- Основные операции над стеком
 - включение нового элемента (английское название push - заталкивать)
 - исключение элемента из стека (англ. pop - выскакивать).



Стек

- Применение стека эффективно, когда нужно реализовать:
 - обмен данными между методами приложения с помощью параметров
 - синтаксический анализ разнообразных выражений
- Реализация стека возможна разными способами:
 - в виде статического массива
 - в виде динамического массива
 - в виде односвязного списка
 - в виде двусвязного списка.


Стек – реализация на основе массива

Класс стека реализует

- поля:

- внутренний массив-указатель на обобщенный тип и переменную, определяющую количество элементов в стеке

- базовые функции (методы) для организации работы стека



Стек – реализация на основе односвязного списка

Преимущества

- меньше объем памяти в случае добавления нового элемента
- меньшее количество дополнительных операций в случае манипулирования стеком (добавление нового элемента, удаление элемента)

Недостатки

- сложность реализации
- для доступа к элементу в динамическом массиве удобно использовать доступ по индексу. В односвязном списке нужно пересматривать весь список от начала к нужной позиции

Очередь FIFO

- **Очередью FIFO** (First – In – First - Out - "первым пришел - первым исключается") называется такой последовательный список с переменной длиной, в котором
 - включение элементов выполняется только с одной стороны списка (эту сторону часто называют концом или хвостом очереди),
 - а исключение - с другой стороны (называемой началом или головой очереди).
- **Основные операции:**
 - включение,
 - исключение,
 - определение размера, очистка,





Способы реализации очереди

На основе:

- Статического массива с ограничением на размер в очереди
- Динамического массива
- Односвязного списка
- Двусвязного списка

Дек

- **Дек** – особый вид очереди
- Дек (deq - double ended queue, т.е очередь с двумя концами) – это такой последовательный список, в котором как включение, так и исключение элементов может осуществляться с любого из двух концов списка
- Операции над деком:
 - ☐ включение элемента справа;
 - ☐ включение элемента слева;
 - ☐ исключение элемента справа;
 - ☐ исключение элемента слева;
 - ☐ определение размера; очистка.

Очереди в Python

- Использование списка для создание очереди FIFO

```
q = []

q.append('eat')
q.append('sleep')
q.append('code')

print(q)
# ['eat', 'sleep', 'code']

# медленно работает!
print(q.pop(0)) # 'eat'
```

Очереди в Python

- Использование класса **deque** модуля **collections**

```
from collections import deque
```

```
q = deque()
```

```
q.append('eat')
```

```
q.append('sleep')
```

```
q.append('code')
```

```
q.appendleft('name')
```

```
print(q)           # deque(['name', 'eat', 'sleep', 'code'])
```

```
print(q.popleft())  # 'name'
```

```
print(q.popleft())  # 'eat'
```

```
print(q.pop())      # 'code'
```

Эта структура данных
позволяет добавлять и
удалять элементы с каждой
стороны со сложностью **$O(1)$**

Очереди в Python

- Использование класса **Queue** модуля **queue**

```
"""FIFO Queue  
"""
```

```
import queue
```

```
q = queue.Queue()
```

```
for i in range(5):  
    q.put(i)
```

```
while not q.empty():  
    print(q.get())
```

Класс реализующий очередь
FIFO

Модуль queue содержит несколько
классов, которые полезны в
параллельных вычислениях

`Queue.put(item, [block[, timeout]])`

Если **block=True** и **timeout** не задан (None по умолчанию), то при необходимости произойдет блокировка до тех пор пока в очереди не будет доступного места

0
1
2
3
4

Очереди в Python

- Использование класса **Queue** модуля **queue**

```
"""LIFO Queue
"""
```

```
import queue
```

```
q = queue.LifoQueue()
```

```
for i in range(5):  
    q.put(i)
```

```
while not q.empty():  
    print(q.get())
```

Класс реализующий очередь
LIFO

4
3
2
1
0

`q.get_nowait()` – метод возвращает элемент очереди немедленно, иначе генерирует исключение `Queue.Empty`

