**ЧАСТЬ ХАЦКЕВИЧ**

1. Анжелика (с 13 по 16 включительно)
2. Вика (с 17 по 20 включительно)
3. Паша (с 21 по 25 включительно)
4. Даник (с 26 по 30 включительно)
5. Саша (с 31 по 35 включительно)
6. Антон (с 36 по 40 включительно)
7. Дима (с 41 по 44 включительно)

**ЧАСТЬ МИХНО**

1. Анжелика (с 1 по 5 включительно)
2. Вика (с 6 по 10 включительно)
3. Паша (с 11 по 15 включительно)
4. Даник (Вероника) (с 16 по 20 включительно)
5. Саша (с 21 по 26 включительно)
6. Дима (с 27 по 31 включительно)



**Вопросы по дисциплине «ЯП»**

**3 семестр**

**ЧАСТЬ ХАЦКЕВИЧ**

**1. Классы С++. Описание класса. Пример. Описание объектов. Пример.Указатель this. Пример.**

**Класс** − фундаментальное понятие С++, он лежит в основе многих свойств С++. Класс предоставляет механизм для создания объектов. В классе отражены важнейшие концепции объектно-ориентированного программирования: инкапсуляция, наследование, полиморфизм.

В простейшем случае класс можно определить с помощью конструкции:

*тип\_класса имя\_класса{список\_членов\_класса}*;

*тип\_класса* – одно из служебных слов class, struct, union;

*имя\_класса* – идентификатор;

*список\_членов\_класса* – определения и описания типизированных данных и принадлежащих классу функций.

**Функции** – это методы класса, определяющие операции над объектом.

Данные – это поля объекта, образующие его структуру. Значения полей определяет состояние объекта.

Конкретные переменные типа данных "класс" называются **экземплярами** класса, или **объектами**.

Для описания объекта класса (экземпляра класса) используется конструкция

имя\_класса имя\_объекта;

monster Vasia; //Объект класса monster с параметрами по умолчанию

monster Super(200, 300);// Объект с явной инициализацией

monster stado[100];// Массив объектов с параметрами по умолчанию

/\* Динамический объект (второй параметр задается по умолчанию) \*/

monster \*beavis = new monster (10);

monster &butthead = Vasia;// Ссылка на объект

Доступ к открытым ( public ) элементам объекта аналогичен доступу к полям структуры. Для этого используются операция . (точка) при обращении к элементу через имя объекта и операция -> при обращении через указатель:

объект.поле

указатель -> поле

(\*указатель).поле

объект.метод( параметры )указатель -> метод( параметры )

(\*указатель).метод( параметры )

Обращение к открытому полю и вызов метода для массива объектов:

имя\_массива[ индекс ].поле

имя\_массива[ индекс ].метод( параметры )

Например:

int n = Vasia.get\_ammo();

stado[5].draw;

cout << beavis->get\_health();

**Указатель this.**

Когда функция-член класса вызывается для обработки данных конкретного объекта, этой функции автоматически и неявно передается указатель на тот объект, для которого функция вызвана. Этот указатель имеет имя this и неявно определен в каждой функции класса следующим образом:

имя\_класса \*const this = адрес\_объекта

Указатель this является дополнительным скрытым параметром каждой нестатической компонентной функции. При входе в тело принадлежащей классу функции this инициализируется значением адреса того объекта,для которого вызвана функция. В результате этого объект становится доступным внутри этой функции.

В большинстве случаев использование this является неявным. В частности, каждое обращение к нестатической функции-члену класса неявно использует this для доступа к члену соответствующего объекта.

Выражение \*this представляет собой разыменование указателя и имеет тип определяемого класса. Обычно это выражение возвращается в качестве результата, если метод возвращает ссылку на свой класс (return \*this;).

monster & the\_best(monster &M){

if( health > M.get\_health()) {

return \*this;

}

return M;

}  
  
monster Vasia(50), Super(200);  
// Новый объект Best инициализируется значениями полей Super  
monster Best = Vasia.the\_best(Super);

**2. Классы С++. Конструкторы. Свойства конструкторов. Конструктор**

**копирования. Пример. Деструкторы.**

**Конструктор**

Для инициализации объектов класса в его определение можно явно включить специальную компонентную функцию, называемую конструктором. Формат определения конструктора следующий:

имя\_класса(список\_форм\_параметров){операторы\_тела\_конструктора}

Имя этой компонентной функции по правилам языка С++ должно совпадать с именем класса. Такая функция автоматически вызывается при определении или размещении в памяти с помощью оператора new каждого объекта класса.

Конструктор выделяет память для объекта и инициализирует данные − члены класса.

**Свойства конструктора:**

• Для конструктора не определяется тип возвращаемого значения. Даже тип void не допустим.

•Указатель на конструктор не может быть определен, и соответственно нельзя получить адрес конструктора.

•Конструкторы не наследуются.

•Конструкторы не могут быть описаны с ключевыми словами virtual, static, const, mutuable, valatile.

Конструктор всегда существует для любого класса, причем, если он не определен явно, он создается автоматически. По умолчанию создается конструктор без параметров и конструктор копирования. Если конструктор описан явно, то конструктор по умолчанию не создается. По умолчанию конструкторы создаются общедоступными (public).

Параметром конструктора не может быть его собственный класс, но может быть ссылка на него (T&).

Существуют два способа инициализации данных объекта с помощью конструктора: передача значений параметров в тело конструктора, список инициализаторов данного класса.

По умолчанию создается также конструктор копирования вида T::T(const T&), где Т – имя класса. Конструктор копирования вызывается всякий раз, когда выполняется копирование объектов, принадлежащих классу. В частности он вызывается:

а) когда объект передается функции по значению;

б) при построении временного объекта как возвращаемого значения функции;

в) при использовании объекта для инициализации другого объекта.

Если класс не содержит явным образом определенного конструктора копирования, то при возникновении одной из этих трех ситуаций производится побитовое копирование объекта. Побитовое копирование не во всех случаях является адекватным. Именно для таких случаев и необходимо определить собственный конструктор копирования.

**Деструктор**

Динамическое выделение памяти для объекта создает необходимость освобождения этой памяти при уничтожении объекта. Желательно, чтобы освобождение памяти происходило автоматически. Такую возможность обеспечивает специальный компонент класса – деструктор класса. Его формат:

~имя\_класса(){операторы\_тела\_деструктора}

Имя деструктора совпадает с именем его класса, но предваряется символом “~” (тильда).

Деструктор не имеет параметров и возвращаемого значения. Вызов деструктора выполняется не явно (автоматически), как только объект класса уничтожается.

Если в классе деструктор не определен явно, то компилятор генерирует деструктор по умолчанию, который просто освобождает память, занятую данными объекта. В тех случаях, когда требуется выполнить освобождение и других объектов памяти, например область, на которую указывает.

Так же, как и для конструктора, не может быть определен указатель на деструктор.

**Примеры:**

Exam::Exam() {

cout << "// Конструктор без параметров " << this << endl;

}

Exam::Exam(char\*name, int date) {

this->name = name;

this->date = date;

cout << "// Конструктор с параметрами " << this << endl;

}

Exam::Exam(const Exam &cpexam) {

student\_name = cpexam.student\_name;

student\_mark = cpexam.student\_mark;

cout << "// Конструктор копирования " << this << endl;

}

Exam::~Exam() {

cout << "// Деструктор" << this << endl;

}

**3. Классы С++. Статические элементы класса. Статические поля. Статические**

**методы. Пример. Статические функции-члены. Указатель this.**

**Статические члены класса.**

Такие компоненты должны быть определены в классе, как статические (static). Статические данные классов не дублируются при создании объектов, т.е. каждый статический компонент существует в единственном экземпляре. Доступ к статическому компоненту возможен только после его инициализации. Для инициализации используется конструкция

тип имя\_класса : : имя\_данного инициализатор;

Например, int complex : : count = 0;

Это предложение должно быть размещено в глобальной области после определения класса. Только при инициализации статическое данное класса получает память и становится доступным. Обращаться к статическому данному класса можно обычным образом через имя объекта

имя\_объекта.имя\_компонента

Но к статическим компонентам можно обращаться и тогда, когда объект класса еще не существует. Доступ к статическим компонентам возможен не только через имя объекта, но и через имя класса

имя\_класса :: имя\_компонента

Однако так можно обращаться только к public компонентам.

Для обращения к private статической компоненте извне можно с помощью статических компонентов-функций. Эти функции можно вызвать через имя класса.

имя\_класса :: имя\_статической\_функции

Пример.

class TPoint {

double x,y;

static int N; // статический компонент − данное: количество точек

public:

TPoint (double x1 = 0.0,double y1 = 0.0){

N++; x = x1; y = y1;

}

static int &count(){ // статический компонент-//функция

return N;

}

};

int TPoint :: N = 0; //инициализация статического //компонента-данного

void main(void){

TPoint A(1.0,2.0);

TPoint B(4.0,5.0);

TPoint C(7.0,8.0);

cout<< \nОпределены ”<<TPoint : : count()<<“точки”;

}

**Статические поля**

Статические поля применяются для хранения данных, общих для всех объектов класса, например, количества объектов или ссылки на разделяемый всеми объектами ресурс. Эти поля существуют для всех объектов класса в единственном экземпляре, то есть не дублируются.

Память под статическое поле выделяется один раз при его инициализации независимо от числа созданных объектов (и даже при их отсутствии) и инициализируется с помощью операции доступа к области действия, а не операции выбора:

class A{

public:  
static int count;

}  
A::count = 0;

Статические поля доступны как через имя класса, так и через имя объекта:  
/\* будет выведено одно и то же \*/  
A \*a, b; \* cout << A::count << a->count << b.count;

Классическое применение статических полей - подсчет объектов. Для этого в классе объявляется целочисленное поле, которое увеличивается в конструкторе и уменьшается в деструкторе.

**Статические методы**

Статические методы могут обращаться непосредственно только к статическим полям и вызывать только другие статические методы класса, поскольку им не передается скрытый указатель this. Обращение к статическим методам производится так же, как к статическим полям - либо через имя класса, либо, если хотя бы один объект класса уже создан, через имя объекта.

Статические методы не могут быть константными ( const ) и виртуальными ( virtual ).

**Указатель this.**

Когда функция-член класса вызывается для обработки данных конкретного объекта, этой функции автоматически и неявно передается указатель на тот объект, для которого функция вызвана. Этот указатель имеет имя this и неявно определен в каждой функции класса следующим образом:

имя\_класса \*const this = адрес\_объекта

Указатель this является дополнительным скрытым параметром каждой нестатической компонентной функции. При входе в тело принадлежащей классу функции this инициализируется значением адреса того объекта,для которого вызвана функция. В результате этого объект становится доступным внутри этой функции.

В большинстве случаев использование this является неявным. В частности, каждое обращение к нестатической функции-члену класса неявно использует this для доступа к члену соответствующего объекта.

Выражение \*this представляет собой разыменование указателя и имеет тип определяемого класса. Обычно это выражение возвращается в качестве результата, если метод возвращает ссылку на свой класс (return \*this;).

monster & the\_best(monster &M){

if( health > M.get\_health()) {

return \*this;

}

return M;

}  
  
monster Vasia(50), Super(200);  
// Новый объект Best инициализируется значениями полей Super  
monster Best = Vasia.the\_best(Super);

**4. Дружественная функция С++. Особенности дружественной функции. Пример.**

**Дружественный класс.**

**Дружественные функции** не являются членами класса, однако имеют доступ к его закрытым членам - переменным и функциям, которые имеют спецификатор private.

Для определения дружественных функций используется ключевое слово friend.

class Auto{

friend void drive(Auto &);

friend void setPrice(Auto &, int price);

public:

Auto(string autoName, int autoPrice){

name = autoName;

price = autoPrice; }

string getName(){ return name; }

int getPrice() { return price; }

private:

string name; // название автомобиля

int price; // цена автомобиля

};

void drive(Auto &a) {

cout << a.name;}

void setPrice(Auto &a, int price){

if (price > 0)

a.price = price;}

int main(){

Auto tesla("Tesla", 5000);

drive(tesla);

cout << tesla.getName() << " : " << tesla.getPrice();

setPrice(tesla, 8000);

cout << tesla.getName() << " : " << tesla.getPrice();

}

Когда мы объявляем дружественные функции, то фактически мы говорим компилятору, что это друзья класса и они имеют доступ ко всем членам этого класса, в том числе закрытым.

При этом для дружественных функций не важно, определяются они под спецификатором public или private. Для них это не имеет значения.

Определение этих функций производится вне класса. И поскольку эти функции являются дружественными, то внутри этих функций мы можем через переданную ссылку Auto обратиться ко всем его закрытым переменным.

Дружественные функции могут определяться в другом классе. Если метод класса A внутри тела класса B объявляется с модификатором friend, что указывает на то, что он является другом класса, то из него разрешен доступ ко всем членам класса B.

Например:

class A { public: int Fx();}

class B { public: friend int A::Fx();

private:

}

Объявление **дружественного класса** позволяет всем его методам получить доступ ко всем переменным и методам другого класса.

Если все функции-члены одного класса являются дружественными функциями второго класса, то можно объявить дружественный класс: friend class ИмяКласса;

class A {...};

class B

{

friend class A;

};

Все функции-члены класса А будут иметь доступ к скрытым членам класса В.

Дружественные классы не наследуются, и их дружественность не является транзитивной.

class A {int Fx();}

class B {friend class A;}

class C {friend class B;}

// Класс А не является дружественным классу C

class D : public B {}

// Класс А не является дружественным классу D

**5. Перегрузка операций С++. Основные правила перегрузки. Перегрузка**

**унарных операций. Пример. Перегрузка бинарных операций. Пример.**

Для того чтобы перегрузить операцию в С++ следует объявить функцию или функцию-член класса следующего вида:

<тип> operator <символ> ([<список параметров>]);

Здесь <тип> - тип результата операции, <символ> - символьное обозначение операции, <список параметров> - параметры функции перегрузки операции.

Если предполагается, что операнд при выполнении операции не изменяется (например, операнды в операциях +, -, &&, ||), то в функцию его лучше передавать по значению или константной ссылке. Причем, если операндом является объект, занимающий большой объем памяти, то его лучше передавать по константной ссылке. Для базовых типов данных вполне подойдет передача по значению.

Если операнд изменяется при выполнении операции (например, первый операнд в операциях +=, &=, ++), то такой операнд должен передаваться по неконстантной ссылке.

Не допускается перегрузка следующих операций:

:: ?: \* . typeid sizeof

Существуют операции, которые могут быть перегружены только с использованием нестатических функций-членов классов. Это операции =, (), [], ->.

Перегрузка операторов основана на следующих **принципах**:

1. Один из аргументов должен иметь тип класс.

2. Перегружать можно только встроеные операторы, новые ввести нельзя.

3. Нельзя изменить приоритет операторов и число их аргументов.

4. Можно изменить смысл оператора.

5. Перегрузка основана на определении методов и функций специального вида

**Бинарный**

Функция operator для перегрузки (доопределения) **бинарных операторов** может быть описана двумя способами:

- как компонента-функция класса с одним аргументом, перегружаемой **внутри** класса:

<тип> operator <символ>( <тип2> <парам2>);

- как глобальная функция (функция, описанная вне класса) с двумя аргументами, перегружаемой **вне** класса:

<тип> operator <символ> (<тип1> <парам1>, <тип2> <парам2>);

При перегрузке бинарного оператора # выражение a#b может быть представлено при первом способе как a.operator#(b) или как operator #(a,b) при втором способе перегрузки.

Пример:

// перегрузка операции сравнения

int operator != (CMyString& s1, CMyString& s2){

return !( s1 == s2 );

}

**Унарный**

Унарный оператор, как и бинарный, может быть перегружен двумя способами:

- как компонента-функция без аргументов, перегружаемой **внутри** класса:

<тип> operator <символ>();

- как глобальная функция с одним аргументом, перегружаемой **вне** :

<тип> operator <символ> (<тип1> <парам1>);

Как известно, унарный оператор может быть префиксным и постфиксным. Для любого префиксного унарного оператора выражение #a может быть представлено при первом способе как a.operator#(), а при втором как #operator(a).

При перегрузке унарного оператора, используемого в постфиксной форме, выражение вида a# может быть представлено при первом способе как a.operator#(int) или как operator#(a,int) при втором способе. При этом аргумент типа int не существует и используется для отличия префиксной и постфиксной форм при перегрузке.

Пример:

// перегрузка операции ++A

void operator++(int x){

x++;

}

**6. Перегрузка операций С++. Операторы, которые нельзя перегружать.**

**Операторы, которые можно перегружать в качестве методов**

Для того чтобы перегрузить операцию в С++ следует объявить функцию или функцию-член класса следующего вида:

<тип> operator <символ> ([<список параметров>]);

Здесь <тип> - тип результата операции, <символ> - символьное обозначение операции, <список параметров> - параметры функции перегрузки операции. Количество параметров в списке зависит от того является ли операция унарной или бинарной, а также от способа перегрузки операции. Перегрузка операторов не изменяет порядок выполнения операций и их приоритет. Унарный оператор не может использоваться для переопределения бинарной операции так же, как и бинарный оператор не переопределит унарную операцию.

-В случае, когда перегрузка выполняется функцией вне пользовательского класса, количество параметров функции совпадает с количеством операндов перегружаемой операции.

-В случае, когда перегрузка выполняется функцией-членом класса количество параметров функции на единицу меньше числа операндов перегружаемой операции. В этом случае первый из операндов операции передается в функцию-член класса в качестве указателя this и отсутствует в списке параметров функции.

Операция, в которой первый аргумент не является объектом пользовательского класса, не может быть перегружена функцией-членом класса, а только внешней по отношению к классу функцией. К сожалению, ввести новое символьное обозначение для операции нельзя. Также нельзя перегрузить оператор для базовых типов данных.

Не допускается перегрузка следующих операций:

:: ?: \* . typeid sizeof

Существуют операции, которые могут быть перегружены только с использованием нестатических функций-членов классов. Это операции =, (), [], ->.

Перегрузка операторов основана на следующих **принципах**:

1. Один из аргументов должен иметь тип класс.

2. Перегружать можно только встроеные операторы, новые ввести нельзя.

3. Нельзя изменить приоритет операторов и число их аргументов.

4. Можно изменить смысл оператора.

5. Перегрузка основана на определении методов и функций специального вида

**7. Перегрузка операций С++. Перегрузка операции вызова функции. Пример.**

**Перегрузка операции присваивания. Пример.**

Для того чтобы перегрузить операцию в С++ следует объявить функцию или функцию-член класса следующего вида:

<тип> operator <символ> ([<список параметров>]);

Здесь <тип> - тип результата операции, <символ> - символьное обозначение операции, <список параметров> - параметры функции перегрузки операции. Количество параметров в списке зависит от того является ли операция унарной или бинарной, а также от способа перегрузки операции. Перегрузка операторов не изменяет порядок выполнения операций и их приоритет. Унарный оператор не может использоваться для переопределения бинарной операции так же, как и бинарный оператор не переопределит унарную операцию.

Перегрузка операторов основана на следующих **принципах**:

1. Один из аргументов должен иметь тип класс.

2. Перегружать можно только встроеные операторы, новые ввести нельзя.

3. Нельзя изменить приоритет операторов и число их аргументов.

4. Можно изменить смысл оператора.

5. Перегрузка основана на определении методов и функций специального вида

**Перегрузка операции присваивания**

Операция присваивания определена в любом классе по умолчанию как поэлементное копирование. Эта операция вызывается каждый раз, когда одному существующему объекту присваивается значение другого. Операция-функция должна возвращать ссылку на объект, для которого она вызвана, и принимать в качестве параметра единственный аргумент - ссылку на присваиваемый объект:

Операцию присваивания можно определять только в теле класса. Она не наследуется.

Пример

Class first\_Class{

private:

    int x;

int y;

public:

// Перегрузка оператора присваивания

first\_Class & operator= (const first\_Class &M){

// Выполняем копирование значений

x = M.x;

y = M.y;

// Возвращаем текущий объект, чтобы иметь возможность связать в цепочку выполнение нескольких операций присваивания

return \*this;

}

};

**Перегрузка операции вызова функции**

Операция присваивания рассматривается как бинарная операция, второй аргумент которой представляет совой список параметров вызова(содержащий ноль или более параметров) . Перегружается операция как функция-член класса.

Пример

class Matrix

{

private:

    double data[5][5];

public:

    Matrix(){

// Присваиваем всем элементам массива значение 0.0

        for (int i = 0; i < 5; ++i)

            for (int j = 0; j < 5; ++j)

                data[i][j] = 0.0;

     }

};

double& Matrix::operator()(int row, int col)

{

    assert(col >= 0 && col < 5);

    assert(row >= 0 && row < 5);

    return data[row][col];

}

**8. Наследование. Ключи доступа. Простое наследование С++. Множественное**

**наследование С++.**

**Наследование** − это механизм получения нового класса на основе уже существующего. Существующий класс может быть дополнен или изменен для создания нового класса.

Существующие классы называются базовыми, а новые – производными. Производный класс наследует описание базового класса; затем он может быть изменен добавлением новых членов, изменением существующих функций-членов и изменением прав доступа. С помощью наследования может быть создана иерархия классов, которые совместно используют код и интерфейсы.

Наследуемые компоненты не перемещаются в производный класс, а остаются в базовых классах.

В иерархии производный объект наследует разрешенные для наследования компоненты всех базовых объектов (public, protected).

Допускается множественное наследование – возможность для некоторого класса наследовать компоненты нескольких никак не связанных между собой базовых классов. В иерархии классов соглашение относительно доступности компонентов класса следующее:

**private** – член класса может использоваться только функциями –членами данного класса и функциями – “друзьями” своего класса. В производном классе он недоступен.

**protected** – то же, что и private, но дополнительно член класса с данным атрибутом доступа может использоваться функциями-членами и функциями – “друзьями” классов, производных от данного.

**public** – член класса может использоваться любой функцией, которая является членом данного или производного класса, а также к public - членам возможен доступ извне через имя объекта.

Следует иметь в виду, что объявление friend не является атрибутом доступа и не наследуется.

Синтаксис определения производного класса:

class имя\_класса : [private | protected | public] список\_базовых\_классов

{список\_компонентов\_класса};

В производном классе унаследованные компоненты получают статус доступа private, если новый класс определен с помощью ключевого слова class, и статус public, если с помощью struct.

Явно изменить умалчиваемый статус доступа при наследовании можно с помощью атрибутов доступа – private, protected и public, которые указываются непосредственно перед именами базовых классов.

Объекты класса конструируются снизу-вверх: сначала базовый, потом компоненты-объекты (если они имеются), а потом сам производный класс. Таким образом, объект производного класса содержит в качестве подобъекта объект базового класса.

Уничтожаются объекты в обратном порядке: сначала производный, потом его компоненты-объекты, а потом базовый объект.

**Простым** называется наследование, при котором производный класс имеет одного родителя. Для различных элементов класса существуют разные правила наследования.

**Множественное** наследование означает, что класс имеет несколько базовых классов. При этом, если в базовых классах есть одноименные элементы, может произойти конфликт идентификаторов, который устраняется с помощью операции доступа к области видимости. Если у базовых классов есть общий предок, это приведет к тому, что производный от этих базовых класс унаследует два экземпляра полей предка, что чаще всего является нежелательным. Чтобы избежать такой ситуации, требуется при наследовании общего предка определить его как виртуальный класс.

**9. Наследование, виртуальные функции и полиморфизм в С++.**

**Наследование** − это механизм получения нового класса на основе уже существующего. Существующий класс может быть дополнен или изменен для создания нового класса.

Существующие классы называются базовыми, а новые – производными. Производный класс наследует описание базового класса; затем он может быть изменен добавлением новых членов, изменением существующих функций-членов и изменением прав доступа. С помощью наследования может быть создана иерархия классов, которые совместно используют код и интерфейсы.

Наследуемые компоненты не перемещаются в производный класс, а остаются в базовых классах.

**Виртуальные функции.**

К механизму виртуальных функций обращаются в тех случаях, когда в каждом производном классе требуется свой вариант некоторой компонентной функции. Классы, включающие такие функции, называются полиморфными и играют особую роль в ООП.

Виртуальные функции предоставляют механизм позднего (отложенного) или динамического связывания. Любая нестатическая функция базового класса может быть сделана виртуальной, для чего используется ключевое слово virtual.

Пример.

class base

{

public:

virtual void print(){cout<<“\nbase”;}

...

};

class dir : public base

{

public:

void print(){cout<<“\ndir”;}

};

void main()

{

base B,\*bp = &B;

dir D,\*dp = &D;

base \*p = &D;

bp –>print(); // base

dp –>print(); // dir

p –>print(); // dir

}

Таким образом, интерпретация каждого вызова виртуальной функции через указатель на базовый класс зависит от значения этого указателя, т.е. от типа объекта, для которого выполняется вызов.

Выбор того, какую виртуальную функцию вызвать, будет зависеть от типа объекта, на который фактически (в момент выполнения программы) направлен указатель, а не от типа указателя.

Виртуальными могут быть только нестатические функции-члены.

Виртуальность наследуется. После того как функция определена как виртуальная, ее повторное определение в производном классе (с тем же самым прототипом) создает в этом классе новую виртуальную функцию,причем спецификатор virtual может не использоваться.

Конструкторы не могут быть виртуальными, в отличие от деструкторов. Практически каждый класс, имеющий виртуальную функцию, должен иметь виртуальный деструктор.

Полиморфизм — это способность обьекта использовать методы производного класса, который не существует на момент создания базового.

**10. Наследование и спецификаторы доступа в С++. Спецификатор доступа принаследовании базового класса. Передача параметров в базовый класс. Ссылкина производные классы**

**Наследование** − это механизм получения нового класса на основе уже существующего. Существующий класс может быть дополнен или изменен для создания нового класса.

Существующие классы называются базовыми, а новые – производными. Производный класс наследует описание базового класса; затем он может быть изменен добавлением новых членов, изменением существующих функций-членов и изменением прав доступа. С помощью наследования может быть создана иерархия классов, которые совместно используют код и интерфейсы.

Наследуемые компоненты не перемещаются в производный класс, а остаются в базовых классах.

**Наследование** – это свойство объектов, благодаря которому экземпляры класса могут иметь доступ к данным и методам, содержащимся в ранее определенном классе, без повторного их определения.

**Спецификаторы доступа:**

В С++ члены класса классифицируются в соответствии с правами доступа на три категории:

– public (публичные) – любая функция программы имеет к ним доступ;

– private (частные) – доступ к ним имеют только функции-члены класса или дружественные функции;

– protected (защищенные) – доступ к ним имеют функции-члены класса или дружественные функции, а также производные классы.

При описании класса в его заголовке перечисляются (через запятую) все классы, которые являются для него базовыми. Возможность обращения к элементам этих классов регулируется с помощью спецификаторов доступа.

**Спецификатор доступа при наследовании базового класса:**

Элементы protected при наследовании с ключом private становятся в производном классе private, в остальных случаях права доступа к ним не изменяются.

Доступ к элементам public при наследовании становится соответствующим ключу доступа.

Если базовый класс наследуется с ключом private, можно выборочно сделать некоторые его элементы доступными в производном классе, объявив их в секции public производного класса с помощью операции доступа к области видимости.

**Передача параметров в базовый класс:**

В случае, когда базовый класс содержит конструктор с аргументами, производные классы должны явно передавать базовому классу необходимые аргументы.

Для этого используется расширенная форма конструкторов производных классов, в которые передаются аргументы конструкторам базового класса.

Эта форма выглядит следующим образом:

порожденый\_конструктор(список\_аргументов):базовый\_класс1(список\_аргументов), базовый\_класс2(список\_аргументов), . . ., базовый\_классN(список\_аргументов)

{

. . . тело класса

};

**Ссылки на производные классы**

Предположим, например, что имеется базовый класс B\_class и его производный класс D\_class. В С++ любой указатель типа B\_class\* может также указывать на объект типа D\_class. Например, если имеются следующие объявления переменных:  
  
B\_class \*р; // указатель на объект типа B\_class  
B\_class B\_ob; // объект типа B\_class  
D\_class D\_ob; // объект типа D\_class

то следующие присвоения абсолютно законны:  
  
р = &В\_оb; // р указывает на объект типа B\_class  
р = &D\_ob; /\* р указывает на объект типа D\_class, являющийся объектом, порожденным от B\_class \*/

**11. Виртуальные функции в С++. Чисто виртуальные функции и абстрактныетипы. Виртуальный базовый класс.**

**Виртуальные функции.**

К механизму виртуальных функций обращаются в тех случаях, когда в каждом производном классе требуется свой вариант некоторой компонентной функции. Классы, включающие такие функции, называются полиморфными и играют особую роль в ООП.

Виртуальные функции предоставляют механизм позднего (отложенного) или динамического связывания. Любая нестатическая функция базового класса может быть сделана виртуальной, для чего используется ключевое слово virtual.

Пример.

class base

{

public:

virtual void print(){cout<<“\nbase”;}

...

};

class dir : public base

{

public:

void print(){cout<<“\ndir”;}

};

void main()

{

base B,\*bp = &B;

dir D,\*dp = &D;

base \*p = &D;

bp –>print(); // base

dp –>print(); // dir

p –>print(); // dir

}

Таким образом, интерпретация каждого вызова виртуальной функции через указатель на базовый класс зависит от значения этого указателя, т.е. от типа объекта, для которого выполняется вызов.

Выбор того, какую виртуальную функцию вызвать, будет зависеть от типа объекта, на который фактически (в момент выполнения программы) направлен указатель, а не от типа указателя.

Виртуальными могут быть только нестатические функции-члены.

Виртуальность наследуется. После того как функция определена как виртуальная, ее повторное определение в производном классе (с тем же самым прототипом) создает в этом классе новую виртуальную функцию,причем спецификатор virtual может не использоваться.

Конструкторы не могут быть виртуальными, в отличие от деструкторов. Практически каждый класс, имеющий виртуальную функцию, должен иметь виртуальный деструктор.

**Абстрактные классы.**

Абстрактным называется класс, в котором есть хотя бы одна чистая(пустая) виртуальная функция.

**Чистой виртуальной функцией** называется компонентная функция, которая имеет следующее определение:

virtual тип имя\_функции (список\_формальных\_параметров) = 0;

Чистая виртуальная функция ничего не делает и недоступна для вызовов. Ее назначение – служить основой для подменяющих ее функций в производных классах. Абстрактный класс может использоваться только в качестве базового для производных классов.

Механизм абстрактных классов разработан для представления общих понятий, которые в дальнейшем предполагается конкретизировать. При этом построение иерархии классов выполняется по следующей схеме. Во главе иерархии стоит абстрактный базовый класс. Он используется для наследования интерфейса. Производные классы будут конкретизировать и реализовать этот интерфейс. В абстрактном классе объявлены чистые виртуальные функции, которые по сути есть абстрактные методы.

Пример.

class Base{

public:

Base(); // конструктор по умолчанию

Base(const Base&); // конструктор копирования

virtual ~Base(); // виртуальный деструктор

virtual void Show()=0; // чистая виртуальная функция

// другие чистые виртуальные функции

protected: // защищенные члены класса

private:

// часто остается пустым, иначе будет мешать будущим //разработкам

};

class Derived: virtual public Base{

public:

Derived(); // конструктор по умолчанию

Derived(const Derived&); // конструктор копирования

Derived(параметры); // конструктор с параметрами

virtual ~Derived(); // виртуальный деструктор

void Show(); // переопределенная //виртуальная функция

// другие переопределенные виртуальные функции

// другие перегруженные операции

protected:

// используется вместо private, если ожидается наследование

private:

// используется для деталей реализации

};

Объект абстрактного класса не может быть формальным параметром функции, однако формальным параметром может быть указатель на абстрактный класс. В этом случае появляется возможность передавать в вызываемую функцию в качестве фактического параметра значение указателя на производный объект, заменяя им указатель на абстрактный базовый класс. Таким образом мы получаем полиморфные объекты.

**Виртуальный базовый класс**

**Виртуальный базовый класс** — это класс, объект которого является общим для использования всеми дочерними классами. Вот пример создания общего родительского класса:

| class PoweredDevice {      PoweredDevice(int power) {  std::cout << "PoweredDevice: " << power << '\n';      }  };    class Scanner: public PoweredDevice{      Scanner(int scanner, int power) : PoweredDevice(power) {  std::cout << "Scanner: " << scanner << '\n';      }  };    class Printer: public PoweredDevice {      Printer(int printer, int power) : PoweredDevice(power) {  std::cout << "Printer: " << printer << '\n';      }  };    class Copier: public Scanner, public Printer {      Copier(int scanner, int printer, int power): Scanner(scanner, power), Printer(printer, power)      {    }  };  Диаграмма наследования:  https://ravesli.com/wp-content/uploads/2018/10/inheriting-diagram.jpg |
| --- |

Теперь, при создании класса Copier, мы получим только одну копию PoweredDevice, которая будет общей как для Scanner, так и для Printer.

**12. Объявление класса в отдельном файле. Отделение интерфейса от реализациив С++.**

Подключение файлов выполняется с помощью препроцессорной директивы #include. Но даже, если мы сможем подключить файл с классом, появится новая проблема — так как в файле с классом уже есть функция main(), то при построении проекта компилятор выдаст ошибку. Суть ошибки: «В проекте найдено несколько main() — функций.» Именно поэтому класс необходимо объявлять в отдельном файле, чтобы его можно было неоднократно использовать. Ранее мы объявляли в отдельном файле функции, таким же образом размещается класс в отдельном файле.

Для этого необходимо выполнить 3 шага:

1)добавить в проект заголовочный файл \*.h;

2)в заголовочном файле объявить пользовательский класс, в нашем случае — CppStudio;

3)подключить заголовочный файл к программе, в нашем случае — #include "CppStudio.h".

**Отделение интерфейса от реализации**

Интерфейс класса — конструкция, определяющая методы и свойства, предоставляемые классом. Реализация класса — это способ осуществления работоспособности класса. До этого мы не отделяли интерфейс класса от его реализации, то есть реализация методов осуществлялась внутри класса. Отделение интерфейса от реализации класса выполняется для того, чтобы скрыть способ осуществления работоспособности класса. Отделение интерфейса от реализации выполняется за 5 шагов:

1)добавить в проект заголовочный файл \*.h;

2)определить интерфейс класса в заголовочном файле;

3)добавить в проект исполняемый файл \*.cpp;

4)в исполняемом файле выполнить реализацию класса;

5)подключить заголовочный файл к программе.

Имена заголовочному и исполняемому файлам даются, как правило, одинаковые.

**13. Информационная система. Данные. База данных. Ведение базы данных. Система управления базами данных. Классификация АИС(автоматизированные информационные системы) по типу хранимых данных**

**Информационная система (ИС)** — система, предназначенная для хранения, поиска и обработки информации, и соответствующие организационные ресурсы (человеческие, технические, финансовые и т. д.), которые обеспечивают и распространяют информацию. **Данные** — формы представления информации, с которыми имеют дело информационные системы и их пользователи**. База данных** — это упорядоченный набор структурированной информации или данных, которые обычно хранятся в электронном виде в компьютерной системе. База данных обычно управляется **системой управления базами данных (СУБД) –** комплексом программ, позволяющих создать базу данных и манипулировать данными (вставлять, обновлять, удалять и выбирать). Система обеспечивает безопасность, надёжность хранения и целостность данных, а также предоставляет средства для администрирования БД. **Ведение базы данных** — деятельность по обновлению, восстановлению и перестройке структуры базы данных с целью обеспечения ее целостности, сохранности и эффективности использования.

**Классификация АИС:**



*Информационно-поисковые системы* производят ввод, систематизацию, хранение, выдачу информации по запросу пользователя без сложных преобразований данных. Например, информационно-поисковая система в библиотеке, в железнодорожных- и авиакассах продажи билетов. *Информационно-решающие системы* осуществляют все операции переработки информации по определенному алгоритму. Среди них можно провести классификацию по степени воздействия выработанной результатной информации на процесс принятия решений и выделить два класса: *управляющие и советующие*. *Управляющие ИС* вырабатывают информацию, на основании которой человек принимает решение. Для этих систем характерны тип задач расчетного характера и обработка больших объемов данных. Примером могут служить система оперативного планирования выпуска продукции, система бухгалтерского учета. *Советующие ИС* вырабатывают информацию, которая принимается человеком к сведению и не превращается немедленно в серию конкретных действий. Эти системы обладают более высокой степенью интеллекта, так как для них характерна обработка знаний, а не данных. *Пример.* Существуют медицинские информационные системы для постановки диагноза больного и определения предполагаемой процедуры лечения. Врач при работе с подобной системой может принять к сведению полученную информацию, но предложить иное по сравнению с рекомендуемым решение.

**14. Уровни представления данных. Трехуровневая архитектура ANSI/SPARC**

**Архитектура ANSI-SPARC** (также **трёхуровневая архитектура**) определяет принцип, согласно которому рекомендуется [строить](https://ru.wikipedia.org/wiki/%D0%90%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B_%D0%B1%D0%B0%D0%B7%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) системы управления базами данных (СУБД).

**Три уровня представления данных:**

1. Внешний (пользовательский) – тот, на котором представляют данные пользователи;

2. Промежуточный (концептуальный) – служит для отображения данных внешнего уровня на внутренний и обеспечивает необходимую независимость данных разных уровней друг от друга;

3. Внутренний (физический) – на внутреннем уровне данные воспринимаются СУБД и операционной системой.

В основе архитектуры ANSI-SPARC лежит концептуальный(2) уровень. В современных СУБД он может быть реализован при помощи логической таблицы. Концептуальный уровень описывает данные и их взаимосвязи с наиболее общей точки зрения, — концепции архитекторов базы, используя реляционную или другую модель.

**15. Трехуровневая архитектура ANSI/SPARC. Внутренняя схема базы данных.**

**Архитектура ANSI-SPARC** (также **трёхуровневая архитектура**) определяет принцип, согласно которому рекомендуется [строить](https://ru.wikipedia.org/wiki/%D0%90%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B_%D0%B1%D0%B0%D0%B7%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) системы управления базами данных (СУБД).

Внутренняя схема определяет физическую структуру хранения базы данных. Внутренняя схема представляет собой низкоуровневое представление всей базы данных. Центральным компонентом внутренней схемы являются информационные массивы, собственно данные и массивы индексов*.* Массивы индексов представляют специальные дополнительные конструкции для ускорения доступа к данным основных информационных объектов. Информационные массивы в большинстве СУБД состоят из одной или нескольких страниц, каждая из которых содержит совокупность некоторых единичных элементов, называемых физическими записями.

**16. Трехуровневая архитектура ANSI/SPARC. Концептуальный уровень.**

**Архитектура ANSI-SPARC** (также **трёхуровневая архитектура**) определяет принцип, согласно которому рекомендуется [строить](https://ru.wikipedia.org/wiki/%D0%90%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B_%D0%B1%D0%B0%D0%B7%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) системы управления базами данных (СУБД).

**Концептуальный уровень –** это объединяющее представление данных, используемых всеми пользовательскими приложениями, работающими с данной базой. На этом уровне база данных представляет собой общий взгляд пользователя на данные проектируемой базы. Это, например, представление с точки зрения менеджера высшего уровня о данных всего предприятия. Здесь описывается: какие данные хранятся в базе, и каковы связи между ними. Этот уровень отражает логическую структуру всей базы с точки зрения администратора базы данных. В действительности концептуальный уровень отражает обобщенную модель предметной области (объектов реального мира), для которой создается база данных.

Концептуальный уровень – это попытка представить требования к базе со стороны организации. И этот уровень не должен содержать никаких сведений о методах хранения данных. Здесь должны быть отражены:

* все сущности, включаемые в базу, их атрибуты и связи;
* накладываемые на данные ограничения;
* семантическая информация о данных;
* информация о мерах обеспечения безопасности и поддержки целостности данных.

База данных на концептуальном уровне имеет высокую степень абстракции и *характеризуется* *аппаратной и программной независимостью*. Создаваемая на этом уровне концептуальная модель служит основой для идентификации и описания основных объектов данных.

**17. Трехуровневая архитектура ANSI/SPARC. Внешний уровень.**

Архитектура ANSI-SPARC (также трёхуровневая архитектура) определяет принцип, согласно которому рекомендуется [строить](https://ru.wikipedia.org/wiki/%D0%90%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B_%D0%B1%D0%B0%D0%B7%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) [системы управления базами данных](https://ru.wikipedia.org/wiki/%D0%A1%D0%A3%D0%91%D0%94) (СУБД).

Проект архитектуры был выдвинут в [1975 году](https://ru.wikipedia.org/wiki/1975) подкомитетом [SPARC](https://ru.wikipedia.org/w/index.php?title=Standards_Planning_and_Requirements_Committee&action=edit&redlink=1) [ANSI](https://ru.wikipedia.org/wiki/%D0%90%D0%BC%D0%B5%D1%80%D0%B8%D0%BA%D0%B0%D0%BD%D1%81%D0%BA%D0%B8%D0%B9_%D0%BD%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%B8%D0%BD%D1%81%D1%82%D0%B8%D1%82%D1%83%D1%82_%D1%81%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82%D0%BE%D0%B2).

Три уровня системы:

1. Внешний (пользовательский)

2. Промежуточный (концептуальный)

3. Внутренний (физический).

На внешнем уровне описываются различные подмножества элементов концептуального уровня для представлений данных различным пользовательским программам. Каждый [пользователь](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C) получает в своё распоряжение часть представлений о данных, но полная концепция скрыта. Отделение внешнего уровня от концепту**ального обеспечивает *логическую независимость* данных.**

**18. Предметная область. Сущность. Атрибут сущности. Экземпляр сущности. Связь. Концептуальная схема базы данных.**

Понятие предметной области базы данных является одним из базовых понятий информатики и не имеет точного определения. Его использование в контексте ИС предполагает существование устойчивой во времени соотнесенности между именами, понятиями и определенными реалиями внешнего мира, не зависящей от самой ИС и ее круга пользователей.

Сущность предметной области является результатом абстрагирования реального объекта путем выделения и фиксации набора его свойств.Примерами сущностей (с точки зрения ИС) или объектов (с точки зрения внешнего мира) являются отдельный студент, группа студентов, аудитория, время занятий, слова, числа, символы. Обычно считается, что быть объектом - это значит быть дискретным и различимым. Примеры "не-объектов" - это мир, время, смысл, хотя и такие категории могут сохраняться в базе данных.

***Атрибут сущности*** - это именованная характеристика, являющаяся некоторым свойством сущности.Примерами атрибутов сущности "Сотрудник" могут быть такие атрибуты как "Табельный номер", "Фамилия", "Имя", "Отчество", "Должность", "Зарплата" и т.п.

***Экземпляр сущности*** - это конкретный представитель данной сущности.

Например, представителем сущности "Сотрудник" может быть "Сотрудник Иванов".

Экземпляры сущностей должны быть *различимы*, т.е. сущности должны иметь некоторые свойства, уникальные для каждого экземпляра этой сущности.

***Связь*** - это некоторая ассоциация между *двумя* сущностями. Одна сущность может быть связана с другой сущностью или сама с собою.

Связи позволяют по одной сущности находить другие сущности, связанные с нею.

Например, связи между сущностями могут выражаться следующими фразами - "СОТРУДНИК может иметь несколько ДЕТЕЙ", "каждый СОТРУДНИК обязан числиться ровно в одном ОТДЕЛЕ".

Концептуальная модель [базы данных](https://webonto.ru/chto-takoe-baza-dannyih/) это некая наглядная диаграмма, нарисованная в принятых обозначениях и подробно показывающая связь между объектами и их характеристиками. Создается концептуальная модель для дальнейшего проектирования базы данных и перевод ее, например, в реляционную базу данных. На концептуальной модели в визуально удобном виде прописываются связи между объектами данных и их характеристиками.

**19. Основные модели данных. Модель данных. Иерархическая модель данных. Пример иерархической базы данных. Достоинства/недостатки.**

Модель базы данных - это тип модели данных, которая определяет логическую структуру базы данных и в корне определяет, каким образом данные могут храниться, организовываться и обрабатываться. Самым популярным примером модели базы данных является [реляционная модель](https://ru.bmstu.wiki/RDBMS_(Relational_Database_Management_System)), которая использует табличный формат.

Различают три основные модели базы данных – это иерархическая, сетевая и реляционная. Эти модели отличаются между собой по способу установления связей между данными.

1. Иерархический подход к организации баз данных. Иерархические базы данных имеют форму деревьев с дугами-связями и узлами-элементами данных. Иерархическая структура предполагала неравноправие между данными – одни жестко подчинены другим. Подобные структуры, безусловно, четко удовлетворяют требованиям многих, но далеко не всех реальных задач.
2. Сетевая модель данных. В сетевых БД наряду с вертикальными реализованы и горизонтальные связи. Однако унаследованы многие недостатки иерархической и главный из них, необходимость четко определять на физическом уровне связи данных и столь же четко следовать этой структуре связей при запросах к базе.
3. Реляционная модель данных. Реляционная модель появилась вследствие стремления сделать базу данных как можно более гибкой. Данная модель предоставила простой и эффективный механизм поддержания связей данных.
4. Объектно-ориентированная модель. Новые области использования вычислительной техники, такие как научные исследования, автоматизированное проектирование и автоматизация учреждений, потребовали от баз данных способности хранить и обрабатывать новые объекты – текст, аудио- и видеоинформацию, а также документы.

Иерархические базы данных — самая ранняя модель представления сложной структуры данных. Информация в иерархической базе организована по принципу древовидной структуры, в виде отношений «предок-потомок». Каждая запись может иметь не более одной родительской записи и несколько подчиненных.

Рассмотрим следующую модель данных предприятия: предприятие состоит из отделов, в которых работают сотрудники. В каждом отделе может работать несколько сотрудников, но сотрудник не может работать более чем в одном отделе.

недостатки иерархических БД:

1. Частично дублируется информация между записями СОТРУДНИК и ИСПОЛНИТЕЛЬ (такие записи называют парными), причем в иерархической модели данных не предусмотрена поддержка соответствия между парными записями.
2. Иерархическая модель реализует отношение между исходной и дочерней записью по схеме 1:N, то есть одной родительской записи может соответствовать любое число дочерних.

**20. Основные модели данных. Модель данных. Сетевая модель. Пример сетевой модели. Достоинства/недостатки.**

Модель базы данных - это тип модели данных, которая определяет логическую структуру базы данных и в корне определяет, каким образом данные могут храниться, организовываться и обрабатываться. Самым популярным примером модели базы данных является [реляционная модель](https://ru.bmstu.wiki/RDBMS_(Relational_Database_Management_System)), которая использует табличный формат.

Различают три основные модели базы данных – это иерархическая, сетевая и реляционная. Эти модели отличаются между собой по способу установления связей между данными.

1. Иерархический подход к организации баз данных. Иерархические базы данных имеют форму деревьев с дугами-связями и узлами-элементами данных. Иерархическая структура предполагала неравноправие между данными – одни жестко подчинены другим. Подобные структуры, безусловно, четко удовлетворяют требованиям многих, но далеко не всех реальных задач.
2. Сетевая модель данных. В сетевых БД наряду с вертикальными реализованы и горизонтальные связи. Однако унаследованы многие недостатки иерархической и главный из них, необходимость четко определять на физическом уровне связи данных и столь же четко следовать этой структуре связей при запросах к базе.
3. Реляционная модель данных. Реляционная модель появилась вследствие стремления сделать базу данных как можно более гибкой. Данная модель предоставила простой и эффективный механизм поддержания связей данных.
4. Объектно-ориентированная модель. Новые области использования вычислительной техники, такие как научные исследования, автоматизированное проектирование и автоматизация учреждений, потребовали от баз данных способности хранить и обрабатывать новые объекты – текст, аудио- и видеоинформацию, а также документы.

Сетевая модель данных определяется в тех же терминах, что и иерархическая. Она состоит из множества записей, которые могут быть владельцами или членами групповых отношений. Связь между записью-владельцем и записью-членом также имеет вид 1:N.

Основное различие этих моделей состоит в том, что в сетевой модели запись может быть членом более чем одного группового отношения.

Иерархическая структура преобразовывается в сетевую следующим образом:

* деревья (a) и (b), показанные на рис. 4.2, заменяются одной сетевой структурой, в которой запись СОТРУДНИК входит в два групповых отношения;
* для отображения типа M:N вводится запись СОТРУДНИК\_КОНТРАКТ, которая не имеет полей и служит только для связи записей КОНТРАКТ и СОТРУДНИК, см. рис. 4.3 (Отметим, что в этой записи может храниться и полезная информация, например, доля данного сотрудника в общем вознаграждении по данному контракту)

**21. Основные модели данных. Модель данных. Реляционная модель данных. Пример реляционной модели базы данных. Достоинства/недостатки.**

Модель данных — это логическое определение объектов, операторов и прочих элементов, в совокупности составляющих абстрактную машину доступа к данным, с которой взаимодействует пользователь.

Различают как минимум следующие модели данных:

· Логические модели

· Физические модели

· Прочие модели

Реляционная модель данных - логическая модель данных, описывающая:

* структуры данных в виде (изменяющихся во времени) наборов отношений;
* теоретико-множественные операции над данными: объединение, пересечение разность и декартово произведение;
* специальные реляционные операции: селекция, проекция, соединение и деление;
* специальные правила, обеспечивающие целостность данных.

В простейшей БД «ЗАКАЗ» могут быть созданы две таблицы: «Заказ» и «Клиент», - связанные между собой

| Достоинства | Недостатки |
| --- | --- |
| · Эта модель данных отображает информацию в наиболее простой для пользователя форме  · Основана на развитом математи-ческом аппарате, который позволяет достаточно лаконично описать основные операции над данными.  · Позволяет создавать языки манипулирования данными не процедурного типа.  · Манипулирование данными на уровне выходной БД и возможность изменения. | · Самый медленный доступ к данным.  · Трудоемкость разработки |

**22.Реляционная модель данных. Сущность. Отношение. Кортеж. Атрибуты. Степень отношения. Кардинальное число отношения. Домен. Тип данных.**

Домен – это множество отдельных допустимых значений данных, которые:

· есть неделимыми (атомарными) для данной модели;

· имеют одинаковый тип.

Атрибуты – это столбцы таблицы (поля таблицы). Атрибуты имеют имена. По имени атрибута осуществляется обращение к таблице.

Кортеж есть множество пар, которые представлены следующим образом:

{ имя\_атрибута, значение }

Кардинальное число – это количество кортежей.

Тип данных есть характеристикой объекта в языке программирования. Таким объектом может выступать переменная, константа и т.п. Тип данных определяет допустимое множество значений, которые может принимать переменная величина или объект.

Отношение - это множество кортежей (не упорядоченный список), соответствующих одной схеме отношения.

Степень отношения – это количество атрибутов.

Сущность – это любой объект в базе данных, который можно выделить исходя из сути предметной области для которой разрабатывается эта база данных. Разработчик базы данных должен уметь правильно определять сущности.

В базе данных книжного магазина можно выделить следующие сущности:

· книга;

· поставщик;

· размещение в магазине.

**23.Реляционная модель данных. Ключи отношений. Типы ключей. Пример применения ключей.**

Перви́чный ключ (*primary key*) — в [реляционной модели данных](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) один из [потенциальных ключей](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%82%D0%B5%D0%BD%D1%86%D0%B8%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BB%D1%8E%D1%87) [отношения](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BD%D0%BE%D1%88%D0%B5%D0%BD%D0%B8%D0%B5_(%D1%80%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C)), выбранный в качестве основного ключа.

Если в [отношении](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BD%D0%BE%D1%88%D0%B5%D0%BD%D0%B8%D0%B5_(%D1%80%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C)) имеется единственный потенциальный ключ, он является и первичным ключом. Если потенциальных ключей несколько, один из них выбирается в качестве первичного, а другие называют «альтернативными».

Типы ключей баз данных

* Foreign Key. FOREIGN KEY (внешний ключ) служит для указания в одной таблице на первичный ключ в другой.
* Unique Key. UNIQUE KEY позволяет однозначно идентифицировать каждую запись в таблице.

Ключи представляют способ идентификации строк в таблице. С помощью ключей мы также можем связывать строки между различными таблицами в отношения.

**24. Реляционная модель данных. Отношение, схема отношения, схема БД. Множество элементов отношения.**

Отношение - это множество кортежей (не упорядоченный список), соответствующих одной схеме отношения.

Схема отношения - это именованное множество пар {имя атрибута – имя домена}

Схема базы данных включает в себя описания содержания, структуры и ограничений [целостности](https://ru.wikipedia.org/wiki/%D0%A6%D0%B5%D0%BB%D0%BE%D1%81%D1%82%D0%BD%D0%BE%D1%81%D1%82%D1%8C_%D0%B1%D0%B0%D0%B7%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), используемые для создания и поддержки [базы данных](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85).

**25. Реляционная модель данных. Свойства реляционных отношений.**

Свойства отношений реляционной модели данных:

· Каждый атрибут отношения имеет уникальное в данном отношении имя.

· Каждый атрибут определен на каком-то одном домене.

· На одном и том же домене может быть определено несколько атрибутов.

· Имя атрибута может совпадать с именем домена.

· Порядок следования атрибутов не устанавливается (атрибуты в определении схемы отношения не упорядочены).

· В отношении нет совпадающих кортежей (каждый кортеж уникален).

· Порядок следования кортежей не устанавливается (кортежи в отношении не упорядочены).

· Отношение имеет имя, которое в схеме базы данных отличается от имен всех других отношений.

· Отношения делятся на сущности и связи.

**26. Реляционная модель данных. Виды реляционных отношений. Связывание таблиц.**

Реляционная модель данных - это способ рассмотрения данных, то есть предписание для способа представления данных (посредством таблиц) и для способа работы с таким представлением (посредством операторов). Она связана с тремя аспектами данных: структурой (объекты), целостностью и обработкой данных (операторы).

*Реляционные базы* данных представляют собой [базы данных](https://www.oracle.com/ru/database/what-is-database.html), которые используются для хранения и предоставления доступа к взаимосвязанным элементам информации. Реляционные базы данных основаны на реляционной модели — интуитивно понятном, наглядном табличном способе представления данных. Каждая строка, содержащая в таблице такой базы данных, представляет собой запись с уникальным идентификатором, который называют *ключом*. Столбцы таблицы имеют атрибуты данных, а каждая запись обычно содержит значение для каждого атрибута, что дает возможность легко устанавливать взаимосвязь между элементами данных.

Она обеспечила стандартный способ представления данных и отправки запросов, которые могли быть использованы в любых приложениях. Разработчики уяснили, что таблицы являются ключевым преимуществом реляционных баз данных, так как обеспечивают интуитивно понятный, эффективный и гибкий способ хранения структурированной информации и получения к ней доступа.

### **Виды и типы связей между таблицами в реляционных базах данных**

Давайте теперь рассмотрим то, **как могут быть связаны таблицы в реляционных базах данных**. Сразу скажу, что всего существует **три вида связей между таблицами баз данных**:

• связь один к одному;

**Связь один к одному** – самая редко встречаемая связь между таблицами. В 97 случаях из 100, если вы видите такую связь, вам необходимо объединить две таблицы в одну.

• связь один ко многим;

**Связь один ко многим в реляционных базах данных** реализуется тогда, когда объекту А может принадлежать или же соответствовать несколько объектов Б, но объекту Б может соответствовать только один объект А.

• связь многие ко многим.

**Связь многие ко многим** реализуется в том случае, когда нескольким объектам из таблицы А может соответствовать несколько объектов из таблицы Б, и в тоже время нескольким объектам из таблицы Б соответствует несколько объектов из таблицы А.

**27. Реляционная модель данных. Целостность данных. Основные требования. Ограничение целостности. Значение NULL.Целостность сущностей. Целостность связей.**

Реляционная модель данных - это способ рассмотрения данных, то есть предписание для способа представления данных (посредством таблиц) и для способа работы с таким представлением (посредством операторов). Она связана с тремя аспектами данных: структурой (объекты), целостностью и обработкой данных (операторы).

В теории баз данных под целостностью понимают свойство соответствия структуры и содержания базы данных предметной области. В реляционной модели данных определяются **два основных требования**, при которых обеспечивается целостность данных: **целостность сущностей** и **целостность ссылок**.

Каждый объект или наблюдение представляется в [реляционной базе](https://wiki.loginom.ru/articles/relational-database.html) как группа взаимосвязанных элементов данных — кортеж некоторого отношения. Требование целостности сущностей заключается в том, чтобы каждый кортеж любого отношения отличался от другого кортежа этого отношения (т.е. любое отношение должно обладать первичным ключом).

В теории реляционных баз данных принято выделять четыре типа ограничений целостности[[2]](https://ru.wikipedia.org/wiki/%D0%A6%D0%B5%D0%BB%D0%BE%D1%81%D1%82%D0%BD%D0%BE%D1%81%D1%82%D1%8C_%D0%B1%D0%B0%D0%B7%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85#cite_note-%D0%94%D0%B5%D0%B9%D1%82_%D0%92%D0%B2%D0%A1%D0%91%D0%94_8-2):353:

* *Ограничением базы данных* называется ограничение на значения, которые разрешено принимать указанной базе данных.
* *Ограничением* [*переменной отношения*](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BD%D0%BE%D1%88%D0%B5%D0%BD%D0%B8%D0%B5_(%D1%80%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C)) называется ограничение на значения, которые разрешено принимать указанной переменной отношения.
* *Ограничением* [*атрибута*](https://ru.wikipedia.org/w/index.php?title=%D0%90%D1%82%D1%80%D0%B8%D0%B1%D1%83%D1%82_(%D1%80%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C)&action=edit&redlink=1) называется ограничение на значения, которые разрешено принимать указанному атрибуту.
* *Ограничение* [*типа*](https://ru.wikipedia.org/wiki/%D0%94%D0%BE%D0%BC%D0%B5%D0%BD_(%D0%B1%D0%B0%D0%B7%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)) представляет собой ни что иное, как определение множества значений, из которых состоит данный тип.

Примером распространённого ограничения уровня переменной отношения является [потенциальный ключ](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%82%D0%B5%D0%BD%D1%86%D0%B8%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BB%D1%8E%D1%87); примером распространённого ограничения уровня базы данных является [внешний ключ](https://ru.wikipedia.org/wiki/%D0%92%D0%BD%D0%B5%D1%88%D0%BD%D0%B8%D0%B9_%D0%BA%D0%BB%D1%8E%D1%87).

Значение NULL указывает на то, что поле может быть пустым.

# **Целостность сущностей**

Первое ограничение целостности касается первичных ключей базовых отношений. В базовом отношении ни один атрибут первичного , ключа не может содержать отсутствующих значений, обозначаемых как NULL.

Пустое значение: Указывает, что значение атрибута в настоящий момент неизвестно или неприемлемо для этого кортежа.

По определению, первичный ключ — это минимальный идентификатор, который используется для уникальной идентификации кортежей. Это значит, что никакое подмножество первичного ключа не может быть достаточным для уникальной идентификации кортежей. Если допустить присутствие NULL в любой части первичного ключа, это равносильно утверждению, что не все его атрибуты необходимы для уникальной идентификации кортежей, что противоречит определению первичного ключа.

## **Ссылочная целостность**

Второе ограничение целостности касается внешних ключей.

Ссылочная целостность - необходимое качество реляционной базы данных, заключающееся в отсутствии в любом её отношении внешних ключей, ссылающихся на несуществующие кортежи.

Если в отношении существует внешний ключ, то значение внешнего ключа должно либо соответствовать значению потенциального ключа некоторого кортежа в его базовом отношении либо внешний ключ должен полностью состоять из значений NULL.

Например, считается допустимым создание записи с информацией о новом сотруднике с указанием NULL вместо номера отделения, в котором этот сотрудник работает. Такая ситуация может иметь место в том случае, когда сотрудник зачислен в штат компании, но еще не приписан к какому-то конкретному отделению.

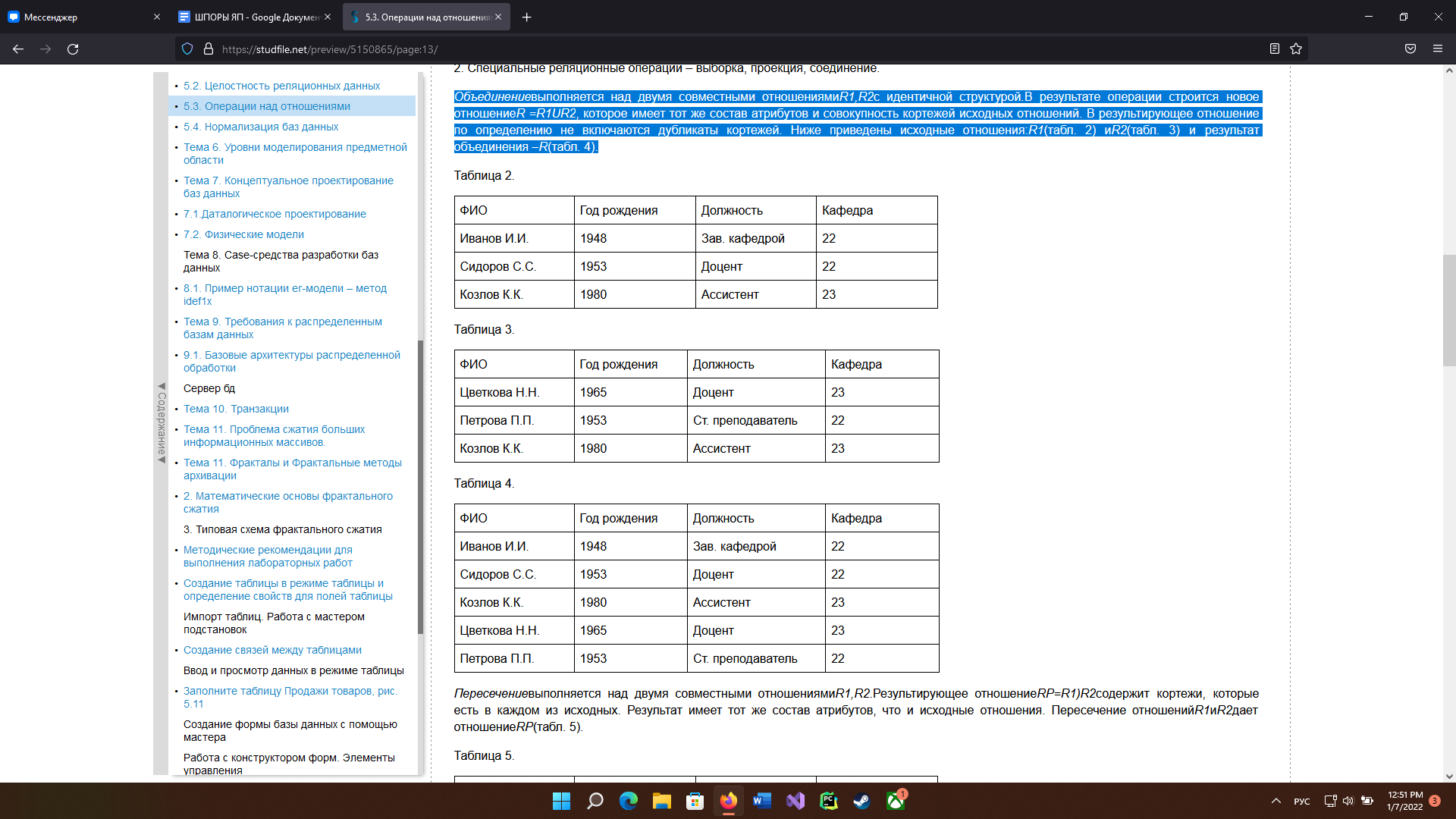
**28. Реляционная модель данных. Операции над отношениями. Объединение. Пример объединения.**

Реляционная модель данных - это способ рассмотрения данных, то есть предписание для способа представления данных (посредством таблиц) и для способа работы с таким представлением (посредством операторов). Она связана с тремя аспектами данных: структурой (объекты), целостностью и обработкой данных (операторы).

**тип данных. Операции над множествами.**

Типа данных задается таким свойством множеств как ограниченность. Для ***однотипных*** элементов появляется возможность сравнения. Над такими однотипными множествами возможны операции:***объединение, пересечение, вычитание (дополнение*), *декартово произведение.***

*Объединение*выполняется над двумя совместными отношениями*R1,R2*с идентичной структурой*.*В результате операции строится новое отношение*R =R1UR2*, которое имеет тот же состав атрибутов и совокупность кортежей исходных отношений. В результирующее отношение по определению не включаются дубликаты кортежей. Ниже приведены исходные отношения:*R1*(табл. 2) и*R2*(табл. 3) и результат объединения –*R*(табл. 4).



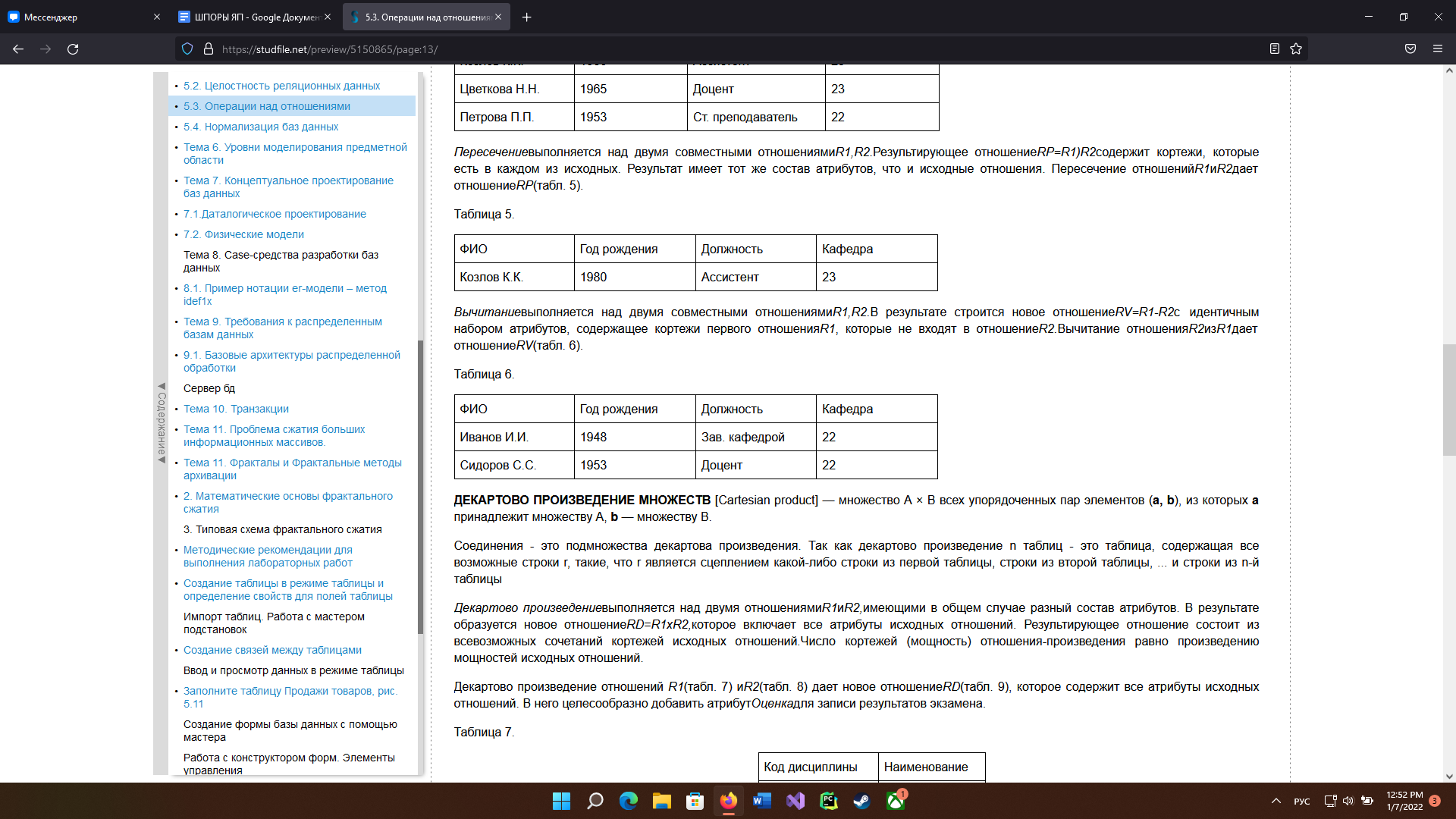
**29. Реляционная модель данных. Операции над отношениями. Пересечение. Пример применения пересечения.**

Реляционная модель данных - это способ рассмотрения данных, то есть предписание для способа представления данных (посредством таблиц) и для способа работы с таким представлением (посредством операторов). Она связана с тремя аспектами данных: структурой (объекты), целостностью и обработкой данных (операторы).

**тип данных. Операции над множествами.**

Типа данных задается таким свойством множеств как ограниченность. Для ***однотипных*** элементов появляется возможность сравнения. Над такими однотипными множествами возможны операции:***объединение, пересечение, вычитание (дополнение*), *декартово произведение.***

*Пересечение*выполняется над двумя совместными отношениями*R1,R2.*Результирующее отношение*RP=R1)R2*содержит кортежи, которые есть в каждом из исходных. Результат имеет тот же состав атрибутов, что и исходные отношения. Пересечение отношений*R1*и*R2*дает отношение*RP*(табл. 5).



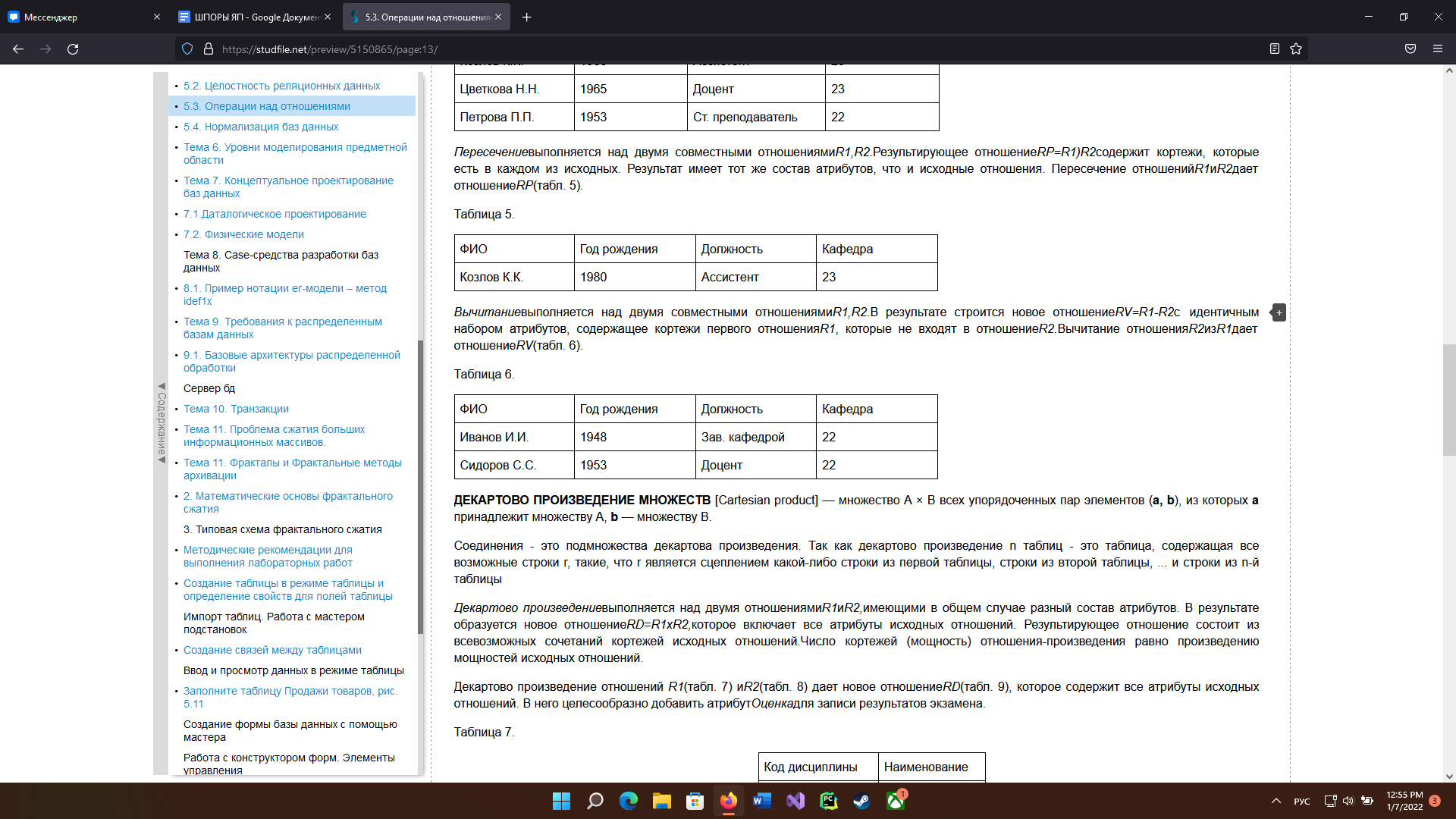
**30. Реляционная модель данных. Операции над отношениями. Вычитание. Пример применения вычитания.**

Реляционная модель данных - это способ рассмотрения данных, то есть предписание для способа представления данных (посредством таблиц) и для способа работы с таким представлением (посредством операторов). Она связана с тремя аспектами данных: структурой (объекты), целостностью и обработкой данных (операторы).

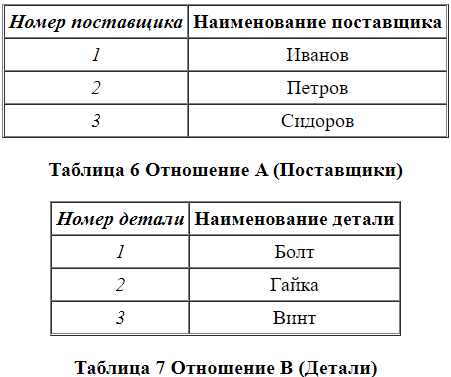
**тип данных. Операции над множествами.**

Типа данных задается таким свойством множеств как ограниченность. Для ***однотипных*** элементов появляется возможность сравнения. Над такими однотипными множествами возможны операции:***объединение, пересечение, вычитание (дополнение*), *декартово произведение.***

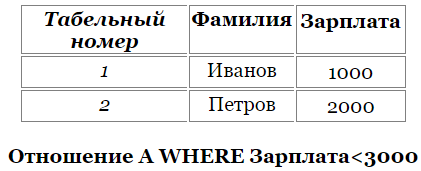
*Вычитание*выполняется над двумя совместными отношениями*R1,R2.*В результате строится новое отношение*RV=R1-R2*с идентичным набором атрибутов, содержащее кортежи первого отношения*R1*, которые не входят в отношение*R2.*Вычитание отношения*R2*из*R1*дает отношение*RV*(табл. 6).



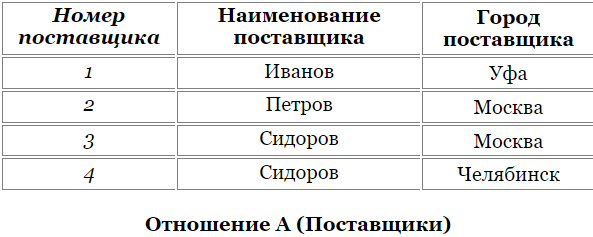
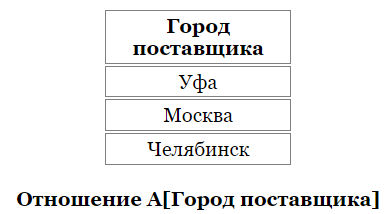
**31. Реляционная модель данных. Операции над отношениями. Декартово произведение. Пример применения декартового произведения.**

Реляционная модель данных - это способ рассмотрения данных, то  
есть предписание для способа представления данных (посредством таблиц) и для способа работы с таким представлением (посредством операторов). Она связана с тремя аспектами данных: структурой (объекты), целостностью и обработкой данных (операторы).  
***Декартовым произведением*** двух отношений  и  называется отношение, заголовок которого является ***сцеплением заголовков*** отношений  и :  
а тело состоит из кортежей, являющихся ***сцеплением кортежей*** отношений  и :, таких, что , .  
Синтаксис операции декартового произведения:  
Пример: пусть даны два отношения  и  с информацией о поставщиках и деталях:  
  
Декартово произведение отношений  и  будет иметь вид:  


**32. Реляционная модель данных. Специальные реляционные операторы. Выборка. Пример применения выборки.**

Реляционная модель данных - это способ рассмотрения данных, то  
есть предписание для способа представления данных (посредством таблиц) и для способа работы с таким представлением (посредством операторов). Она связана с тремя аспектами данных: структурой (объекты), целостностью и обработкой данных (операторы).  
Специальные реляционные операторы: выборка, проекция, соединение, деление.  
Выборкой (ограничением, селекцией) на отношении A с условием C называется отношение с тем же заголовком, что и у отношения A, и телом, состоящем из кортежей, значения атрибутов которых при подстановке в условие C дают значение ИСТИНА. C представляет собой логическое выражение, в которое могут входить атрибуты отношения A и (или) скалярные выражения.  
Синтаксис операции выборки: A WHERE C, или A WHERE XY;  
Пример: пусть дано отношение A с информацией о сотрудниках:  
  
Результат выборки *A WHERE Зарплата<3000* будет иметь вид:  


**33. Реляционная модель данных. Специальные реляционные операторы. Проекция. Пример применения проекция.**

Реляционная модель данных - это способ рассмотрения данных, то  
есть предписание для способа представления данных (посредством таблиц) и для способа работы с таким представлением (посредством операторов). Она связана с тремя аспектами данных: структурой (объекты), целостностью и обработкой данных (операторы).  
Специальные реляционные операторы: выборка, проекция, соединение, деление.  
Проекцией отношения A по атрибутам X, Y, …, Z, где каждый из атрибутов принадлежит отношению A, называется отношение с заголовком (X, Y, …, Z) и телом, содержащим множество кортежей вида (x, y, …, z), таких, для которых в отношении A найдутся кортежи со значением атрибута X равным x, значением атрибута Y равным y, …, значением атрибута Z равным z.  
Синтаксис операции проекции: *A[X, Y, …, Z];*Пример: пусть дано отношение A с информацией о поставщиках, включающих наименование и месторасположение:  
  
Проекция *A*[город поставщика] будет иметь вид:  
 **34. Реляционная модель данных. Специальные реляционные операторы. Соединение. Пример применения соединения.**

Реляционная модель данных - это способ рассмотрения данных, то  
есть предписание для способа представления данных (посредством таблиц) и для способа работы с таким представлением (посредством операторов). Она связана с тремя аспектами данных: структурой (объекты), целостностью и обработкой данных (операторы).  
Специальные реляционные операторы: выборка, проекция, соединение, деление.  
Обычно рассматривается несколько разновидностей операции соединения:

* Общая операция соединения
* -соединение (тэта-соединение)
* Экви-соединение
* Естественное соединение

Наиболее важным из этих частных случаев является операция естественного соединения. Все разновидности соединения являются частными случаями общей операции соединения.  
**Общая операция соединения** *Соединением* отношений  и  по условию  называется отношение



 представляет собой логическое выражение, в которое могут входить атрибуты отношений  и  и (или) скалярные выражения.

Таким образом, операция соединения есть результат последовательного применения операций декартового произведения и выборки. Если в отношениях  и  имеются атрибуты с одинаковыми наименованиями, то перед выполнением соединения такие атрибуты необходимо переименовать.  
**Тэта-соединение**

Пусть отношение  содержит атрибут , отношение  содержит атрибут , а  - один из операторов сравнения ( и т.д.). Тогда -***соединением*** отношения  по атрибуту  с отношением  по атрибуту  называют отношение

  
Это частный случай операции общего соединения.  
Иногда, для операции -соединения применяют следующий, более короткий синтаксис:

  
**Экви-соединение**Наиболее важным частным случаем -соединения является случай, когда  есть просто равенство.

Синтаксис *экви-соединения*:

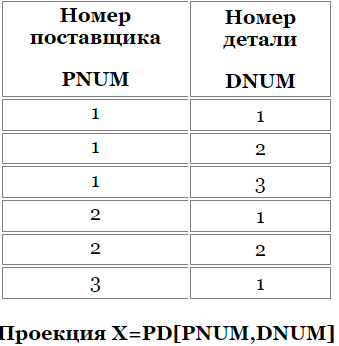
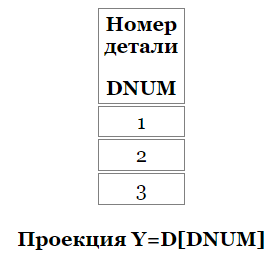
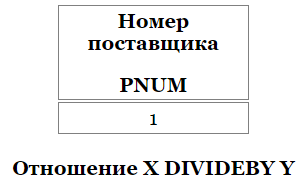
  
**Естественное соединение**Пусть даны отношения  и , имеющие одинаковые атрибуты  (т.е. атрибуты с одинаковыми именами и определенные на одинаковых доменах).

Тогда *естественным соединением* отношений  и  называется отношение с заголовком  и телом, содержащим множество кортежей , таких, что  и .

Естественное соединение настолько важно, что для него используют специальный синтаксис:



**35. Реляционная модель данных. Специальные реляционные операторы. Деление. Пример применения деления.**

Реляционная модель данных - это способ рассмотрения данных, то  
есть предписание для способа представления данных (посредством таблиц) и для способа работы с таким представлением (посредством операторов). Она связана с тремя аспектами данных: структурой (объекты), целостностью и обработкой данных (операторы).  
Специальные реляционные операторы: выборка, проекция, соединение, деление.  
Пусть даны отношения *A(X1, X2, …, Xn, Y1, Y2, …, Ym)* и *B(Y1, Y2, …, Ym),* причем атрибуты *(Y1, Y2, …, Ym)* - общие для двух отношений. Делением отношений A на B называется отношение с заголовком *(X1, X2, …, Xn)* и телом, содержащим множество кортежей *(x1, x2, …, xn)*, таких, что для *всех* кортежей *(y1, y2, …, ym)B* в отношении *A* найдется кортеж *(x1, x2, …, xn, y1, y2, …, ym)*.  
Отношение *A* выступает в роли делимого, отношение *B* выступает в роли делителя. Деление отношений аналогично делению чисел с остатком.  
Синтаксис операции деления: *A DIVIDEBY B;*Пример: в примере с поставщиками, деталями и поставками ответим на вопрос, "какие поставщики поставляют все детали?".  
  
В качестве делителя возьмем проекцию *Y = D[DNUM]*, содержащую список номеров *всех* деталей (не обязательно поставляемых кем-либо):  
  
Деление *X DIVIDEBY Y* дает список номеров поставщиков, поставляющих *все* детали:  
  
Оказалось, что только поставщик с номером 1 поставляет все детали.  
**36. Язык SQL. Свойства таблиц. Ограничения на содержимое таблиц. Основы синтаксиса языка SQL.**

SQL — это язык структурированных запросов (Structured Query Language), позволяющий хранить, манипулировать и извлекать данные из реляционных баз данных

Ограничения — это правила, применяемые к данным. Они используются для ограничения данных, которые могут быть записаны в таблицу. Это обеспечивает точность и достоверность данных в БД.

Ограничения могут устанавливаться как на уровне колонки, так и на уровне таблицы.

Среди наиболее распространенных ограничений можно назвать следующие:

NOT NULL — колонка не может иметь нулевое значение

DEFAULT — значение колонки по умолчанию

UNIQUE — все значения колонки должны быть уникальными

PRIMARY KEY — первичный или основной ключ, уникальный идентификатор записи в текущей таблице

FOREIGN KEY — внешний ключ, уникальный идентификатор записи в другой таблице (таблице, связанной с текущей)

CHECK — все значения в колонке должны удовлетворять определенному условию

INDEX — быстрая запись и извлечение данных

Любое ограничение может быть удалено с помощью команды ALTER TABLE и DROP CONSTRAINT + название ограничения. Некоторые реализации предоставляют сокращения для удаления ограничений и возможность отключать ограничения вместо их удаления.

**37. Язык SQL. Типы данных SQL.**

Наиболее часто используемые типы данных (MySQL):

#### Числовые типы

* INT — целочисленные значения от −2147483648 до 2147483647, 4 байта.
* DECIMAL — хранит числа с заданной точностью. Использует два параметра — максимальное количество цифр всего числа (precision) и количество цифр дробной части (scale). Рекомендуемый тип данных для работы с валютами и координатами. Можно использовать синонимы NUMERIC, DEC, FIXED.
* TINYINT — целые числа от −127 до 128, занимает 1 байт хранимой памяти.
* BOOL — 0 или 1. Однозначный ответ на однозначный вопрос — false или true. Название столбцов типа boolean часто начинается с is, has, can, allow. По факту это даже не отдельный тип данных, а псевдоним для типа TINYINT (1). Тип настолько востребован на практике, что для него в MySQL создали встроенные константы FALSE (0) или TRUE (1). Можно использовать синоним BOOLEAN.
* FLOAT — дробные числа с плавающей запятой (точкой).

#### Символьные

* VARCHAR(N) — N определяет максимально возможную длину строки. Создан для хранения текстовых данных переменной длины, поэтому память хранения зависит от длины строки. Наиболее часто используемый тип строковых данных.
* CHAR(N) — как и с varchar, N указывает максимальную длину строки. Char создан хранить данные строго фиксированной длины, и каждая запись будет занимать ровно столько памяти, сколько требуется для хранения строки длиной N.
* TEXT — подходит для хранения большого объема текста до 65 KB, например, целой статьи.

#### Дата и время

* DATE — только дата. Диапазон от 1000-01-01 по 9999-12-31. Подходит для хранения дат рождения, исторических дат, начиная с 11 века. Память хранения — 3 байта.
* TIME — только время — часы, минуты, секунды — «hh:mm:ss». Память хранения — 3 байта.
* DATETIME — соединяет оба предыдущих типа — дату и время. Использует 8 байтов памяти.
* TIMESTAMP — хранит дату и время начиная с 1970 года. Подходит для большинства бизнес-задач. Потребляет 4 байта памяти, что в два раза меньше, чем DATETIME, поскольку использует более скромный диапазон дат.

#### Бинарные

Используются для хранения файлов, фото, документов, аудио и видеоконтента. Все это хранится в бинарном виде.

* BLOB — до 65 КБ бинарных данных
* LARGEBLOB — до 4 ГБ.

**38. Язык SQL. Создание таблицы. Описание столбца таблицы. Описание первичного ключа. Описание вторичного ключа. Практические примеры.**

Для создания таблиц используется оператор *CREATE TABLE*, который в общем виде выглядит следующим образом:

CREATE [TEMPORARY] TABLE [IF NOT EXISTS]

имя\_таблицы (определение таблицы)

[TYPE=тип\_таблицы];

Ключевое слово *TEMPORARY* используется для создания таблиц, которые будут существовать только в текущем сеансе работы с БД и будут автоматически удалены, когда сеанс завершится.

При использовании выражения *IF NOT EXISTS* таблица будет создана толь­ко в том случае, если еще нет таблицы с указанным именем.

Создать таблицу с такой же схемой, как у существующей, позволяет команда

CREATE [TEMPORARY] TABLE [IF NOT EXISTS]

имя\_таблицы LIKE имя\_старой\_таблицы;

После имени таблицы в скобках объявляются имена столбцов, их типы и другая информация. В определение столбца можно добавить следующие описания:

* Объявить для любого столбца *NOT NULL* или *NULL* (столбцу запрещено или не запрещено содержать значения *NULL*). По умолчанию – *NULL*.
* Объявить для столбца значение по умолчанию, используя ключевое слово *DEFAULT*, за которым должно следовать значение по умолчанию.
* Использовать ключевое слово *AUTO\_INCREMENT*, чтобы генерировать порядковый номер. Автоматически генерируемое значение будет на единицу большим, чем наибольшее значение в таблице. Первая введенная строка будет иметь порядковый номер 1. В таблице можно иметь не более одного столбца *AUTO\_INCREMENT*, и он должен индексироваться.
* Объявить столбец первичным ключом таблицы с помощью выражения *PRIMARY KEY*.
* Объявить столбец внешним ключом, используя выражение *FOREIGN KEY*, со ссылкой на соответствующую таблицу с помощью выражения *REFERENCES*.
* Индексировать столбец с помощью слов *INDEX* или *KEY* (синонимы). Такие столбцы не обяза­тельно должны содержать уникальные значения.
* Индексировать столбец с помощью слова *UNIQUE,* которое используется для указания того, что столбец должен содержать уникальные значения.
* Создать полнотекстовые индексы на основе столбцов типа *TEXT, CHAR* или *VARCHAR* с помощью слова *FULLTEXT* (только с таблицами *MyISAM*).

После закрывающей скобки можно указать тип таблицы:

* *MyISAM* – таблицы этого типа являются «родными» для MySQL, работают очень быстро и поддерживают полнотекстовую индексацию;
* *InnoDB* – ACID-совместимый механизм хранения, поддерживающий транзакции, внешние ключи, каскадное удаление и блокировки на уровне строк;
* *BDB (Berkeley DB)* – является механизмом хранения, который обеспечивает поддержку транзакций и блокировки на уровне страниц;
* *MEMORY (HEAP)* – таблицы целиком хранятся в оперативной памяти и никогда не записываются на диск, поэтому работают очень быстро, но ограничены в размерах и не допускают возможности восстановления в случае отказа системы;
* *MERGE* – тип позволяет объединить несколько таблиц *MyISAM* с одной структурой, чтобы к ним можно было направлять запросы как к одной таблице;
* *NDB Cluster* – тип предназначен для организации кластеровMySQL, когда таблицы распределены между несколькими компьютерами, объединенными в сеть;
* *ARCHIVE* – тип введен для хранения большого объема данных в сжатом формате; таблицы поддерживают только два SQL-оператора: *INSERT* и *SELECT,* причем оператор *SELECT* выполняется по методу полного сканирования таблицы;
* *CSV* – формат представляет собой обычный текстовый фал, записи в котором хранятся в строках, а поля разделены точкой с запятой (широко распространен в компьютерном мире, любая программа, поддерживающая CSV-формат, может открыть такой файл);
* *FEDERATED* – тип позволяет хранить данные в таблицах на другой машине сети (при создании таблицы в локальной директории создается только файл определения структуры таблицы, а все данные хранятся на удаленной машине).

Практический пример:

CREATE TABLE catalogs (

cat\_ID int(6) NOT NULL AUTO\_INCREMENT,

cat\_name varchar(20) NOT NULL,

PRIMARY KEY (cat\_ID)

) TYPE=InnoDB;

CREATE TABLE books (

book\_ID int(6) NOT NULL AUTO\_INCREMENT,

b\_name varchar(100) NOT NULL,

b\_author varchar(100) NOT NULL,

b\_year year NOT NULL,

b\_price decimal(7,2) NULL default '0.00',

b\_count int(6) NULL default '0',

b\_cat\_ID int(6) NOT NULL default '0',

PRIMARY KEY (book\_ID),

FOREIGN KEY (b\_cat\_ID) REFERENCES catalogs(cat\_ID) ON DELETE CASCADE ON UPDATE CASCADE

) TYPE=InnoDB;

**39. Язык SQL. Модификация таблицы. Добавление нового столбца. Удаление первичного ключа. Изменение/удаление значения по умолчанию. Практические примеры.**

Если таблица уже была ранее создана, и ее необходимо изменить, то для этого применяется команда ALTER TABLE. Ее сокращенный формальный синтаксис:

ALTER TABLE название\_таблицы

{ ADD название\_столбца тип\_данных\_столбца [атрибуты\_столбца] |

DROP COLUMN название\_столбца |

MODIFY COLUMN название\_столбца тип\_данных\_столбца [атрибуты\_столбца] |

ALTER COLUMN название\_столбца SET DEFAULT значение\_по\_умолчанию |

ADD [CONSTRAINT] определение\_ограничения |

DROP [CONSTRAINT] имя\_ограничения }

**Добавление нового столбца**

Синтаксис добавления колонки в таблицу:

ALTER TABLE table\_name ADD column\_name data\_type attributes;

Добавим в таблицу Customers новый столбец Address:

ALTER TABLE Customers

ADD Address VARCHAR(50) NULL;

В данном случае столбец Address имеет тип VARCHAR и для него определен атрибут NULL.

**Добавление и удаление первичного ключа**

Добавим в таблицу Products первичный ключ:

CREATE TABLE Products

(

Id INT,

Model VARCHAR(20)

);

ALTER TABLE Products

ADD PRIMARY KEY (Id);

Теперь удалим первичный ключ:

ALTER TABLE Products

DROP PRIMARY KEY;

**Изменение/удаление значения по умолчанию**

С помощью команды ALTER также можно изменить значение по умолчанию для любой колонки, даже если оно не было установлено при создании таблицы.

Синтаксис установки значения по умолчанию для колонки:

ALTER TABLE table\_name ALTER column\_name SET DEFAULT value;

Установим для колонки discount значение по умолчанию:

ALTER TABLE books ALTER discount SET DEFAULT 0;

Чтобы удалить значение по умолчанию, используйте синтаксис:

ALTER TABLE table\_name ALTER column\_name DROP DEFAULT;

Удалим созданное значение по умолчанию для колонки discount:

ALTER TABLE books ALTER dicount DROP DEFAULT;

**40. Язык SQL. Модификация таблицы. Удаление таблицы. Добавление строк в таблицу. Добавление строк в таблицу с использованием списка имен столбцов. Добавление строк в таблицу по результатам запроса к БД. Практические примеры.**

Модификация таблицы в вопросе выше.

Удалить таблицу можно с помощью оператора:

DROP TABLE [IF EXISTS] имя\_таблицы*;*

Например:

DROP TABLE books;

**41. Язык SQL. Модификация таблицы. Выборка данных из таблиц. Оператор SELECT. Описание.**

Для модификации (изменения) отдельных объектов базы данных в SQL языке есть команды и основанные на них sql запросы: SQL ALTER TABLE. Применяется эта команда (запрос), когда нужно сохранить данные в таблицы, и лишь исправить (модифицировать) некоторые из них.

№ 1 Добавляем столбец в таблицу

ALTER TABLE client(client\_site varchar(10));

№ 2 Меняем размер поля столбца

ALTER TABLE client(client\_passwd varchar(25));

№ 3 Меняем размер поля столбца с использованием modify

ALTER TABLE client modify client\_passwd varchar(25);

№ 4 Добавляем столбец, после определенного столбца

ALTER TABLE client add client\_site varchar(50) after client\_telefon;

## **SELECT для выбора столбцов таблицы**

Запрос с оператором SELECT для выбора всех столбцов таблицы имеет следующий синтаксис:

SELECT \* FROM ИМЯ\_ТАБЛИЦЫ

**42. Язык SQL. Арифметические функции. Строковые функции. Операторы и функции, возвращающие логическое значение. Практические примеры.**

**Следующие скалярные функции выполняют вычисление, обычно на основании входных значений, заданных в качестве аргументов, и возвращают числовые значения:**

* [**ABS**](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/abs-transact-sql?view=sql-server-ver15)
* [**ACOS**](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/acos-transact-sql?view=sql-server-ver15)
* [**ASIN**](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/asin-transact-sql?view=sql-server-ver15)
* [**ATAN**](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/atan-transact-sql?view=sql-server-ver15)
* [**ATN2**](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/atn2-transact-sql?view=sql-server-ver15)
* [**CEILING**](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/ceiling-transact-sql?view=sql-server-ver15)
* [**COS**](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/cos-transact-sql?view=sql-server-ver15)
* [**COT**](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/cot-transact-sql?view=sql-server-ver15)
* [**DEGREES**](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/degrees-transact-sql?view=sql-server-ver15)
* [**EXP**](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/exp-transact-sql?view=sql-server-ver15)
* [**FLOOR**](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/floor-transact-sql?view=sql-server-ver15)
* [**LOG**](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/log-transact-sql?view=sql-server-ver15)
* [**LOG10**](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/log10-transact-sql?view=sql-server-ver15)
* [**PI**](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/pi-transact-sql?view=sql-server-ver15)
* [**POWER**](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/power-transact-sql?view=sql-server-ver15)
* [**RADIANS**](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/radians-transact-sql?view=sql-server-ver15)
* [**RAND**](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/rand-transact-sql?view=sql-server-ver15)
* [**ROUND**](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/round-transact-sql?view=sql-server-ver15)
* [**SIGN**](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/sign-transact-sql?view=sql-server-ver15)
* [**SIN**](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/sin-transact-sql?view=sql-server-ver15)
* [**SQRT**](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/sqrt-transact-sql?view=sql-server-ver15)
* [**SQUARE**](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/square-transact-sql?view=sql-server-ver15)
* [**TAN**](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/tan-transact-sql?view=sql-server-ver15)

Примечание

Арифметические функции, такие как ABS, CEILING, DEGREES, FLOOR, POWER, RADIANS и SIGN, возвращают значение того же типа, что и входное значение. Тригонометрические и другие функции, включая EXP, LOG, LOG10, SQUARE и SQRT, преобразуют входные значения в тип float и возвращают значение типа float.

Все математические функции, кроме RAND, являются детерминированными. Это значит, что они возвращают одни и те же результаты каждый раз, когда вызываются с одними и теми же входными значениями. Функция RAND является детерминированной только в том случае, если задан параметр начального значения.

Начнем с функции ABS (x), которая возвращает абсолютное значение переданного ей числа *x*. Пример:

SELECT ABS (-5)

Результат: 5

Начнем с функции FLOOR (x). Возвращает ближайшее целое число не превышающее *x*. Пример:

SELECT FLOOR (5.5)

Результат: 5

Функция MOD (x, y) возвращает остаток от деления *x* на *y*. Например:

SELECT MOD (10, 3)

Результат: 1

Следующая функция EXP (x), которая возвращает значение *e* (Число Эйлера) возведенное в степень *x*. Или научным языком, возвращает экспоненту числа. Пример:

SELECT EXP (3)

Результат: 20.085536923187668

и т.д.

**43. Язык SQL. Модификация таблицы. Упорядочивание и группирование строк результирующей таблицы. Выборка из нескольких таблиц. Практические примеры.**

Для модификации (изменения) отдельных объектов базы данных в SQL языке есть команды и основанные на них sql запросы: SQL ALTER TABLE. Применяется эта команда (запрос), когда нужно сохранить данные в таблицы, и лишь исправить (модифицировать) некоторые из них.

## **Что может выполнить команда ALTER**

С помощью использования команды ALTER можно:

* Добавить столбец в таблицу;
* Добавить ограничение целостности;
* Изменить тип данных в столбце таблицы, его размер, значение по умолчанию (переопределить столбец);
* Удалить столбец;
* Включить, выключить, удалить триггер или ограничение целостности.

№ 1 Добавляем столбец в таблицу

ALTER TABLE client(client\_site varchar(10));//Добавить в таблицу client столбец client\_site//

№ 2 Меняем размер поля столбца

ALTER TABLE client(client\_passwd varchar(25));//Изменение размера поля столбца client\_passwd//

Оператор SQL GROUP BY служит для распределения строк - результата запроса - по группам, в которых значения некоторого столбца, по которому происходит группировка, являются одинаковыми. Группировку можно производить как по одному столбцу, так и по нескольким.

SELECT ИМЕНА\_СТОЛБЦОВ

FROM ИМЯ\_ТАБЛИЦЫ

[WHERE УСЛОВИЕ]

GROUP BY ИМЕНА\_СТОЛБЦОВ

запись в разделе FROM через запятую названий нескольких таблиц позволяет получить результат их декартова произведения (сочетание всех строк "каждая с каждой"). Например, для таблиц Т1 и Т2 это будет выглядеть как

SELECT \*

FROM T1, T2

Добавив в раздел WHERE условие соединения таблиц, можно из декартова произведения получить 9-соединение. Ниже используется соединение по условию равенства значений столбцов Id в таблицах Т1 и Т2:

SELECT \*

FROM Tl, Т2

WHERE Tl.Id=T2. Id

Подобным образом можно соединить и более двух таблиц. В общем случае формат может быть:

SELECT <таблица>.<столбец> [,... п]

FROM <таблица1>, <таблица2> [,... п]

[WHERE <таблица1>.<столбец1>=

<таблица2>.<столбец2>

[AND <условие2> [...]]]

Альтернатива такому способу записи соединения появилась в стандарте ANSI SQL-92 [16| – это инструкция JOIN. Для соединения двух таблиц предлагается следующий синтаксис:

SELECT <таблица>.<столбец> [,... п]

FROM <таблица1>

(INNER | (LEFT I RIGHT I FULL} [OUTER] I CROSS } JOIN

<таблица2>

ON <условие>

**44. Язык SQL. Модификация таблицы. Удаление строк. Изменение содержимого строк.**

Для модификации (изменения) отдельных объектов базы данных в SQL языке есть команды и основанные на них sql запросы: SQL ALTER TABLE. Применяется эта команда (запрос), когда нужно сохранить данные в таблицы, и лишь исправить (модифицировать) некоторые из них.

## **Что может выполнить команда ALTER**

С помощью использования команды ALTER можно:

* Добавить столбец в таблицу;
* Добавить ограничение целостности;
* Изменить тип данных в столбце таблицы, его размер, значение по умолчанию (переопределить столбец);
* Удалить столбец;
* Включить, выключить, удалить триггер или ограничение целостности.

№ 1 Добавляем столбец в таблицу

ALTER TABLE client(client\_site varchar(10));//Добавить в таблицу client столбец client\_site//

№ 2 Меняем размер поля столбца

ALTER TABLE client(client\_passwd varchar(25));//Изменение размера поля столбца client\_passwd//

**ALTER TABLE products DROP COLUMN price, type\_id;**

**ЧАСТЬ МИХНО**

**1. Основные типы данных. Неизменяемые типы**

В Python есть несколько стандартных **типов данных:**

* Numbers (числа)
* Strings (строки)
* Lists (списки)
* Dictionaries (словари)
* Tuples (кортежи)
* Sets (множества)
* Boolean (логический тип данных)

Объекты, значение которых неизменяемо после их создания, называются **неизменяемыми**. Изменчивость объекта определяется его типом. К **неизменяемым (*immutable*) типам** относятся: целые числа (*int*), числа с плавающей точкой (*float*), комплексные числа (*complex*), логические переменные (*bool*), кортежи (*tuple*), строки (*str*) и неизменяемые множества (*frozen set*).

Пример использования неизменяемого типа данных string:

In [14]: intf = 'interface'

In [15]: tun = 'Tunnel0'

In [16]: intf + tun

Out[16]: 'interfaceTunnel0'

In [17]: intf + ' ' + tun

Out[17]: 'interface Tunnel0'

**2. Числовые типы в Python. Библиотека random**

В Python поддерживаются 3 разных числовых типа:

— целые числа со знаком (int);

— значения с плавающей запятой (float);

— комплексные числа (complex).

Для хранения числовых значений используют числовые типы данных. Создание числовых объектов:

var1 **=** 1

var2 **=** 10

Ссылка на числовой объект без проблем удаляется с помощью оператора del. Синтаксис: **del** var1[,var2[,var3[**....**,varN]]]]

У чисел с плавающей запятой ограниченная точность. Визуально разница видна в консоли по наличию точки: 1.0 — число с плавающей запятой, 1 — целое число. Что касается комплексных чисел, то они записываются в виде x+yj. Х в данном случае — действительная часть числа, Y — мнимая.

**Модуль random** предоставляет функции для генерации случайных чисел, букв, случайного выбора элементов последовательности.

*Примеры использования методов модуля:*

**random.sample**(population, k) - список длиной k из последовательности population.

**random.random**() - случайное число от 0 до 1.

**3. Библиотека math. Основные константы и операции**

**Библиотека Math** в Python обеспечивает доступ к некоторым популярным математическим функциям и константам, которые можно использовать в коде для более сложных математических вычислений.

1. Первой важной математической **константой** является число Пи (π). Оно обозначает отношение длины окружности к диаметру. Пример:

import math

radius = 2

print('Площадь окружности с радиусом 2 равна:', math.pi \* (radius \*\* 2))

Площадь окружности с радиусом 2 равна: 12.5………

2. Число Эйлера (е) является основанием натурального логарифма. Получить доступ к числу можно через math.e.

3. Функция log() возвращает логарифм определенного числа.

4. Метод log10() возвращает логарифм по основанию 10 определенного числа:

import math

# Возвращает log10 числа 50

print("log10 числа 50 равен:", math.log10(50))

**Операции:**

ceil(): округление определенного числа вверх;

fabs(): возвращает модуль (абсолютное значение) указанного числа;

floor(): округление определенного числа вниз;

**4. Строки в Python. Основные операции. Особенности**

**Строки** – объекты, состоящие из последовательности символьных данных.

1**. +** — оператор сложения строк. Он возвращает строку, состоящую из других строк, как показано здесь:

>>> s = 'py'

>>> t = 'th'

>>> u = 'on'

>>> s + t

'pyth'

>>> s + t + u

'python'

2**.\*** — оператор создает несколько копий строки.

>>> s = 'py.'

>>> s \* 4

'py.py.py.py.'

>>> 4 \* s

'py.py.py.py.'

3**.** Оператор **in** возвращает True, если подстрока входит в строку, и False, если нет:

>>> s = 'Python'

>>> s in 'I love Python.'

True

>>> s in 'I love Java.'

False

Одной из **особенностей** языка Python является оператор форматирования строк %. Пример:

print ("Меня зовут %s и мой вес равен %d кг!" % ('AndreyEx', 71))

Меня зовут AndreyEx и мой вес равен 71 кг! //вывод

**5. Списки в Python. Основные операции. Вложение, виды вложений**

**Списки** в Python - упорядоченные изменяемые коллекции объектов произвольных типов. Чтобы использовать списки, их нужно создать. Создать список можно несколькими способами. Например, можно обработать любой итерируемый объект (например, строку) встроенной функцией **list**:

**>>>** list('список')

['с', 'п', 'и', 'с', 'о', 'к'] или при помощи литерала:

**>>>** s = [] *# Пустой список*

**>>>** l = ['s', 'p', ['isok'], 2]

**>>>** s

[]

**>>>** l

['s', 'p', ['isok'], 2]

список может содержать любое количество любых объектов (в том числе и вложенные списки), или не содержать ничего.

**Операции:**

| **list.append**(x) | Добавляет элемент в конец списка |
| --- | --- |
| **list.extend**(L) | Расширяет список list, добавляя в конец все элементы списка L |
| **list.insert**(i, x) | Вставляет на i-ый элемент значение x |

| **list.count**(x) | Возвращает количество элементов со значением x |
| --- | --- |

**Пример вложенных списков:**

array = [ [2,4,6], [8,10,12], [14,16,18,20] ]

print([b for a in array for b in a])

*Вывод:*

[2, 4, 6, 8, 10, 12, 14, 16, 18, 20] // один список был создан с использованием двумерного массива, содержащего три списка.

**2 Пример:**

array = [[a for a in range(4)] for b in range(6)]

print(array)

*Вывод:*

[[0, 1, 2, 3], [0, 1, 2, 3], [0, 1, 2, 3], [0, 1, 2, 3], [0, 1, 2, 3], [0, 1, 2, 3]] //двумерный массив создается с помощью функции range(). Команда range() используется для возврата последовательности, которая по умолчанию начинается с 0;

**6 Словари в Python. Основные операции. Вложение, виды вложений**

**Словари в Python** - неупорядоченные коллекции произвольных объектов с доступом по ключу. Их иногда ещё называют ассоциативными массивами или хеш-таблицами.

Перед тем как начать работу со словарем, его нужно создать. Сделать это можно базовыми средствами языка, присвоив свободной переменной произвольное количество пар объектов. Элементы необходимо поместить в фигурные скобки, а между ключом и значением должен стоять символ двоеточия.Вывести содержимое словаря можно стандартной функцией *print,* указав для нее в качестве аргумента нужный набор данных. Для заполнения словаря также используется метод *dict*, получающий произвольное количество пар ключей и значений.Если словарь содержит лишнюю информацию, от нее можно легко избавиться при помощи специальной операции *del*.Функция len позволяет в любой момент определить текущее количество элементов словаря. **Вложенные словари** – это словари, содержащие другие словари. Мы можем создать вложенный словарь так же, как мы создаем обычный словарь, используя фигурные скобки. *Пример вложения словаря*:

**# вложенный словарь, содержащий информацию об известных произведениях искусства**

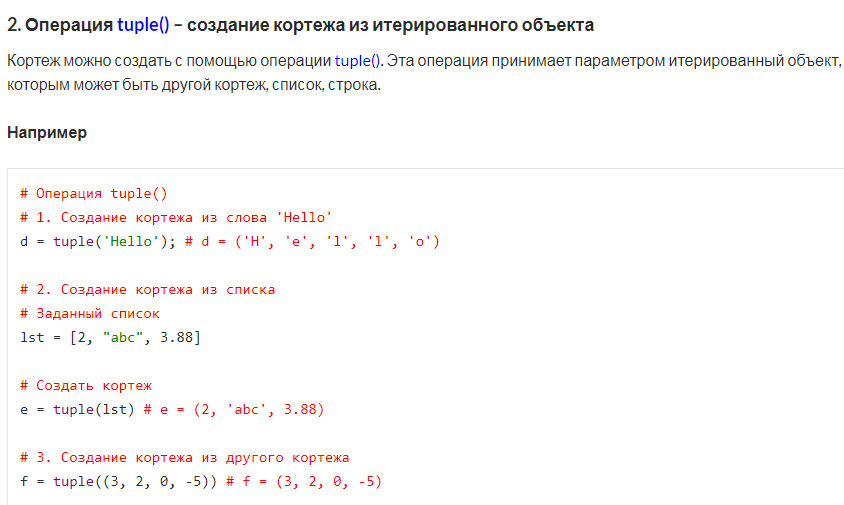
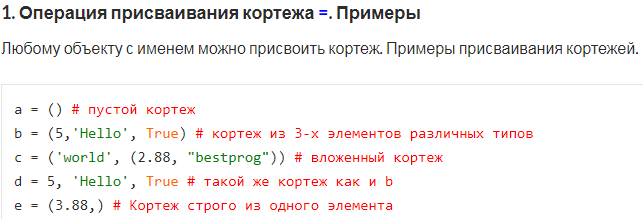
works\_of\_art = {'The\_Starry\_Night': {'author': 'Van Gogh', 'year': 1889, 'style': 'post-impressionist'}, 'The\_Birth\_of\_Venus': {'author': 'Sandro Botticelli', 'year': 1480, 'style': 'renaissance'}}

Создадим вложенный словарь, используя dict(), передавая пары ключ-значение в качестве именованных аргументов:

**# вложенный словарь, созданный при помощи dict()**

works\_of\_art = dict(The\_Starry\_Night={'author': 'Van Gogh', 'year': 1889, 'style': 'post-impressionist'}, The\_Birth\_of\_Venus={'author': 'Sandro Botticelli', 'year': 1480, 'style': 'renaissance'})

**7 Кортежи в Python. Основные операции. Особенности**

Кортеж — это неизменяемый список. Кортеж не может быть изменён никаким способом после его создания. Кортеж определяется так же, как список, за исключением того, что набор элементов заключается в круглые скобки, а не в квадратные.

3. Операция сложения(Конкатенация)

# Конкатенация двух кортежей

A = (1, 2, 3)

B = (4, 5, 6)

C = A + B # C = (1, 2, 3, 4, 5, 6)

**Особенности** – кортежи и списки могут содержать другие составные элементы (строки, списки, кортежи), поддерживают произвольное количество вложений, позволяют сохранять массивы ссылок на другие сложные объекты, которые, в свою очередь также могут быть кортежами, списками или строками.

**8 Множества в Python. Основные операции. Особенности**

**Множество** в python - "контейнер", содержащий не повторяющиеся элементы в случайном порядке.

С множествами можно выполнять множество операций: находить объединение, пересечение и т.д., например

* len(s) - число элементов в множестве (размер множества).
* x in s - принадлежит ли x множеству s.
* **set.copy**() - копия множества.

И операции, непосредственно изменяющие множество, например

* **set.clear**() - очистка множества.
* **set.add**(elem) - добавляет элемент в множество.
* **set.remove**(elem) - удаляет элемент из множества. KeyError, если такого элемента не существует.

**9 Файлы в Python. Основные операции. Unicode и двоичные файлы**

Прежде, чем работать с файлом, его надо открыть. С этим замечательно справится встроенная функция open:

f = open('text.txt', 'r')

У функции open много параметров, они указаны в статье "[Встроенные функции](https://pythonworld.ru/osnovy/vstroennye-funkcii.html)", нам пока важны 3 аргумента: первый, это имя файла. Путь к файлу может быть относительным или абсолютным. Второй аргумент, это режим, в котором мы будем открывать файл.

| Режим | Обозначение |
| --- | --- |
| 'r' | открытие на чтение (является значением по умолчанию). |
| 'w' | открытие на запись, содержимое файла удаляется, если файла не существует, создается новый. |
| 'x' | открытие на запись, если файла не существует, иначе исключение. |
| 'a' | открытие на дозапись, информация добавляется в конец файла. |
| 'b' | открытие в двоичном режиме. |
| 't' | открытие в текстовом режиме (является значением по умолчанию). |
| '+' | открытие на чтение и запись |

Режимы могут быть объединены, то есть, к примеру, 'rb' - чтение в двоичном режиме. По умолчанию режим равен 'rt'.

И последний аргумент, encoding, нужен только в текстовом режиме чтения файла. Этот аргумент задает кодировку.

**10 Типизация в Python. Понятие динамической типизации**

Динами́ческая типиза́ция — приём, используемый в [языках программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) и [языках спецификаци](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D1%81%D0%BF%D0%B5%D1%86%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D0%B9)и, при котором [переменная](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D0%B0%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) связывается с [типом](https://ru.wikipedia.org/wiki/%D0%A2%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) в момент [присваивания](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D1%81%D0%B2%D0%B0%D0%B8%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) [значения](https://ru.wikipedia.org/wiki/%D0%97%D0%BD%D0%B0%D1%87%D0%B5%D0%BD%D0%B8%D0%B5), а не в момент объявления переменной. Таким образом, в различных участках [программы](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0) одна и та же переменная может принимать [значения](https://ru.wikipedia.org/wiki/%D0%97%D0%BD%D0%B0%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) разных [типов](https://ru.wikipedia.org/wiki/%D0%A2%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85).

>>> **class** **Meter**:

... **def** **\_\_len\_\_**(self):

... **return** 1\_000

...

>>> len([1, 2, 3])

3

>>> len("Duck typing...")

14

>>> len(Meter())

1000

В примере выше функции len не важен тип аргумента, а важно лишь то, что у объекта можно вызвать метод \_\_len\_\_().

Но именно эта гибкость и усложняет раннее обнаружение ошибок типизации. Корректность использования объектов определяется динамически, в момент выполнения программы, и зачастую тестирование – единственный способ отловить подобные ошибки. Статическая проверка типов и корректности программы в данном случае представляет значительную сложность.

**11 Операторы в Python. Перечислить и описать**

Операторы Python бывают 7 типов:

· Арифметические операторы

1) Сложение(+)

2) Вычитание(-)

3) Умножение (-)

4) Деление(/)

5) Возведение в степень(\*\*)

6) Деление без остатка(//)

7) Деление по модулю(остаток от деления) (%)

· Операторы сравнения

1) Больше

2) Меньше

3) Больше или равно(>=) меньше или равно(<=)

4) Равно (==)

5) Не равно(!=)

· Операторы присваивания. Оператор присваивания присваивает значение переменной

1) Присваивание(=)

2) Сложение и присваивание(+=)

3) Вычитание и присваивание(-=)

4)Деление и присваивание(/=)

5) Умножение и присваивание(\*=)

6) Деление с остатком и присваивание(//=)

7) Возведение в степень и присваивание(\*\*=)

8) Деление по модулю и присваивание(%=)

· Логические операторы

1) И(and). Если условия с двух сторон оператора and истинны, тогда все выражение целиком считается истинным.

2) ИЛИ(or). Выражение ложно, если оба операнда с двух сторон ложные. Если хотя бы одно из них истинное, то и все выражение истинно.

3) НЕ(not). Этот оператор инвертирует [булевые значения](https://pythonru.com/osnovy/tipy-dannyh-v-python) выражения. True превращается в False и наоборот.

· Операторы принадлежности. Эти операторы проверяют, является ли значение частью последовательности.

1) В(in). Проверяет, является ли значение членом последовательности

>>> pets=['dog','cat', 'ferret']

>>> 'fox' in pets

2) НЕТ В(not in). Этот оператор проверяет, НЕ является ли значение членом последовательности.

· Операторы тождественности. Эти операторы проверяют, являются ли операнды одинаковыми

1) ЭТО(is). Если операнды тождественны, то вернется True

2) ЭТО НЕ(is not). 2 — это число, а '2' — строка. Поэтому вернется True

· Битовые операторы. Эти операторы работают над операндами бит за битом.

1) Бинарное И(&).Проводит побитовую операцию and над двумя значением. Здесь бинарная 2 — это 10, а 3 — 11. Результатом побитового and является 10 — бинарная 2.

2) Бинарное ИЛИ(|).Проводит побитовую операцию and над двумя значением. Здесь бинарная 2 — это 10, а 3 — 11. Результатом побитового and является 10 — бинарная 2.

3) Бинарное ИЛИ НЕТ(^).Проводит побитовую операцию xor (исключающее или) на двух значениях. Здесь результатом ИЛИ НЕ для 10(2) и 11(3) будет 01(1).

4) Инвертирующий оператор(~).Он возвращает инвертированные двоичные числа. Другими словами, переворачивает биты. Битовая 2 — это 00000010. Ее инвертированная версия — 11111101. Это бинарная -3. Поэтому результат -3.

5) Бинарный сдвиг влево(<<).Он сдвигает значение левого операнда на позицию, которая указана справа. Так, бинарная 2 — это 10. 2 << 2 сдвинет значение на две позиции влево и выйдет 1000 — это бинарная 8.

6) Бинарный сдвиг вправо(>>).двигает значение левого оператора на позицию, указанную справа. Так, бинарная 3 — это 11. 3 >> 2 сдвинет значение на два положение вправо. Результат — 00, то есть 0.

**12 Функции в Python. Назначение и применение.**

Функция в python – это составная инструкция,которая может принимать данные ввода,выполнять указания и возвращать данные вывода.

Функции исп-ся для вычислений,для выполнения определённых операций.Очень важный момент – многократное использование кода,что упрощает работу программиста.

Синтаксис:

Def имя\_функции (параметры):

Определение\_функции

**13 Декораторы в Python. Применение и особенности**

Декоратор — это функция, которая позволяет обернуть другую функцию для расширения её функциональности без непосредственного изменения её кода.

Декоратор меняет поведение других функций. При этом он не прерывает работу основной функции. Декораторы в Python могут быть как функциями, так и классами. Обычно декораторы вызываются перед определением функции, которую нужно декорировать.

**14 Аргументы функций. Варианты передачи. Аннотирование типов**

Функция может принимать произвольное количество аргументов или не принимать их вовсе.

Также распространены функции с произвольным числом аргументов, функции с позиционными и именованными аргументами, обязательными и необязательными. Функция также может принимать переменное количество позиционных аргументов, тогда перед именем ставится \*.

Если в функцию передается неизменяемый объект, то этот объект передается «по значению». Это значит, что изменение этого объекта внутри функции не изменит его в вызывающем коде.

Если в функцию передается изменяемый объект, то этот объект передается по «указателю». Изменение такого объекта внутри функции повлияет на этот объект в вызывающем коде.

Аннотирование типов нужно для того чтобы повысить информативность исходного кода, и иметь возможность с помощью сторонних инструментов производить его анализ. Одной из наиболее востребованных, в этом смысле, тем является контроль типов переменных.

**15 Генераторы в Python. Применение и особенности**

Генератор — это объект, который сразу при создании не вычисляет значения всех своих элементов.

· Он хранит в памяти только последний вычисленный элемент, правило перехода к следующему и условие, при котором выполнение прерывается.

· Вычисление следующего значения происходит лишь при выполнении метода next(). Предыдущее значение при этом теряется.

Генераторы используют, чтобы оперативная память не давилась большими объёмами информации. В Python это фишки, экономящие память.

Чтобы создать генератор, необходимо определить функцию, как обычно, но использовать yield вместо return.

**16 Классы в Python. Стандартные методы классов.**

**Классы предоставляют** [**средства объединения данных и функциональности вместе**](https://docs-python.ru/tutorial/klassy-jazyke-python/)**. Создание нового класса создает новый тип объекта, позволяя создавать новые экземпляры этого типа. К каждому экземпляру класса могут быть прикреплены атрибуты для поддержания его состояния. Экземпляры класса также могут иметь методы, определяемые его классом, для изменения его состояния.**

**По сравнению с другими языками программирования, механизм классов Python добавляет классы с минимумом нового синтаксиса и семантики.**

**Классы Python предоставляют все стандартные возможности объектно-ориентированного программирования:**

* [**механизм наследования классов**](https://docs-python.ru/tutorial/klassy-jazyke-python/nasledovanie-klassov/) **позволяет использовать несколько базовых классов,**
* **производный класс может переопределять любые методы своего** [**базового класса**](https://docs-python.ru/tutorial/klassy-jazyke-python/opredelenie-klassov/) **или классов,**
* **метод может вызывать** [**метод базового класса**](https://docs-python.ru/tutorial/klassy-jazyke-python/metod-ekzempljara-klassa/) **с тем же именем.**
* **объекты могут содержать произвольные объемы и типы данных.**
* **как и в случае с** [**модулями**](https://docs-python.ru/tutorial/sistema-importa-python/opredelenie-modulja-import/)**, классы имеют динамическую природу Python: они создаются во время выполнения и могут быть изменены после создания.**

**Простейшая форма определения класса выглядит так:**

**class ClassName:**

**<statement-1>**

**.**

**.**

**.**

**<statement-N>**

**Для определения** [**класса**](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) **используется оператор class:**

**class имя\_класса(надкласс1, надкласс2, ...):**

***# определения атрибутов и методов класса***

**У класса могут быть базовые (родительские) классы (надклассы), которые, если они есть, указываются в скобках после имени определяемого класса.**

**Минимально возможное определение класса выглядит так:**

**class A:**

**pass**

**В терминологии Python члены класса называются атрибутами, функции класса —** [**методами**](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5))**, а** [**поля класса**](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D0%B5_%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D0%B0) **—** [**свойствами**](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D0%BE%D0%B9%D1%81%D1%82%D0%B2%D0%BE_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) **(или просто атрибутами).**

**Определения методов аналогичны определениям функций, но (за некоторыми исключениями, о которых ниже) методы всегда имеют первый аргумент, называемый по общепринятому соглашению self:**

**class A:**

**def m1(self, x):**

***# блок кода метода***

**Определения** [**атрибутов**](https://ru.wikipedia.org/wiki/%D0%90%D1%82%D1%80%D0%B8%D0%B1%D1%83%D1%82_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) **— это обычные операторы присваивания, которые связывают некоторые значения с именами атрибутов.**

**class A:**

**attr1 = 2 \* 2**

**В языке Python класс не является чем-то статическим, поэтому добавить атрибуты можно и после определения:**

**class A:**

**pass**

**def my\_method(self, x):**

**return x \* x**

**A.m1 = my\_method**

**A.attr1 = 2 \* 2**

**Для создания** [**объекта**](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) **— экземпляра класса (то есть, инстанцирования класса), достаточно вызвать класс по имени и задать параметры конструктора:**

**class Point:**

**def \_\_init\_\_(self, x, y, z):**

**self.coord = (x, y, z)**

**def \_\_repr\_\_(self):**

**return "Point(%s, %s, %s)" % self.coord**

**>>> p = Point(0.0, 1.0, 0.0)**

**>>> p**

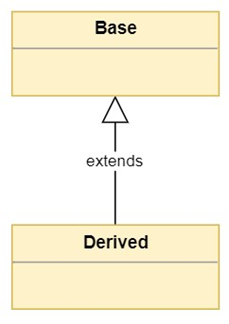
**Point(0.0, 1.0, 0.0)**

**17 Наследование, композиция, агрегация в Python**

**Наследование и композиция являются двумя основными понятиями в объектно-ориентированном программировании, которые моделируют отношения между двумя классами. Они определяют дизайн приложения и определяют, как приложение должно развиваться по мере добавления новых функций или изменения требований. Оба они реализуют повторное использование кода, но делают это по-разному. Наследование предполагает принадлежность к какой-то общности (похожесть), а композиция — формирование целого из частей.**

**Модели наследования — это отношения. Это означает, что когда у вас есть класс Derived, который наследуется от базового класса, вы создали отношение, в котором Derived является специализированной версией Base.**

**Наследование представляется с использованием** [**Unified Modeling Language**](https://www.uml.org/) **или UML следующим образом:**

****

**Классы представлены в виде блоков с именем класса вверху. Отношения наследования представлены стрелкой из производного класса, указывающей на базовый класс. Так же обычно добавляется слово extends к стрелке.**

**Примечание: в отношениях наследования:**

* **Классы, которые наследуются от другого, называются производными классами, подклассами или подтипами.**
* **Классы, из которых получены другие классы, называются базовыми классами или суперклассами.**
* **Считается, что производный класс порождает, наследует или расширяет базовый класс.**

**Родительский класс помещается в скобки после имени класса.**

| **1** | **class FruitTree(Tree):** |
| --- | --- |

**Таким образом, наш код для класса фруктового дерева будет следующим:**

[**?**](http://pythonicway.com/education/python-oop-themes/21-python-inheritance)

| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8** | **class FruitTree(Tree):**  **def \_\_init\_\_(self, kind, height):**  **# Необходимо вызвать метод инициализации родителя.**  **# В Python 3.x это делается при помощи функции super()**  **super().\_\_init\_\_(kind, height)**    **def give\_fruits(self):**  **print ("Collected 20kg of {}s".format(self.kind))** |
| --- | --- |

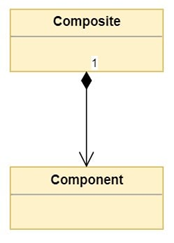
**Создадим экземпляр класса FruitTree:**

| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8** | **tree\_2 = FruitTree("apple", 0.7)**  **# у нас есть доступ к методам родителя**  **tree\_2.info()**  **tree\_2.grow()**  **# Мы можем использовать свой метод**  **tree\_2.give\_fruits()**  **# А для родительского экземпляра метод give\_fruits() недоступен**  **tree\_1.give\_fruits() # Вызовет ошибку** |
| --- | --- |

**§**

**Композиция — это концепция, которая моделирует отношения. Она позволяет создавать сложные типы, комбинируя объекты других типов. Это означает, что класс Composite может содержать объект другого класса Component.**

**UML представляет композицию следующим образом:**

****

**Композиция представляется через линию с ромбом в классе составных элементов, указывающем на класс компонентов. Класс Composite может содержать показатель кардинальности отношений. Кардинальность отношения определяет количество объектов в классе-источнике, которые могут быть связаны с объектами в классе-адресате.**

**На приведенной выше диаграмме 1 означает, что класс Composite содержит один объект типа Component. Кардинальность может быть выражена следующими способами:**

* **Число указывает количество экземпляров Component, которые содержатся в Composite.**
* **Символ \* указывает, что класс Composite может содержать переменное число экземпляров Component.**
* **Диапазон 1..4 указывает, что класс Composite может содержать диапазон экземпляров Component. Диапазон указывается с минимальным и максимальным количеством экземпляров, или с минимальным и множеством экземпляров, как в 1 .. \*.**

**Примечание. Классы, содержащие объекты других классов, обычно называются композитами (composites), а классы, используемые для создания более сложных типов, называются компонентами (components).**

**Например, наш класс Horse может быть составлен другим объектом типа Tail. Композиция позволяет выразить эти отношения, сказав, что у Horse есть Tail.**

**Композиция позволяет повторно использовать код, добавляя объекты к другим объектам, в отличие от наследования интерфейса и реализации других классов. Классы Horse и Dog могут использовать функциональность Tail посредством композиции, не выводя один класс из другого.**

**Рассмотрим на примере реализацию композиции в Python. Пусть, требуется написать программу, которая вычисляет площадь обоев для оклеивания помещения. При этом окна, двери, пол и потолок оклеивать не надо.**

**По условию задачи обои клеятся только на стены, следовательно площади верхнего и нижнего прямоугольников нам не нужны. Из рисунка видно, что площадь одной стены равна xz, второй – уz. Противоположные прямоугольники равны, значит общая площадь четырех прямоугольников равна S = 2xz + 2уz = 2z(x+y). Потом из этой площади надо будет вычесть общую площадь дверей и окон, поскольку они не оклеиваются. Можно выделить три типа объектов – окна, двери и комнаты. Получается три класса. Окна и двери являются частями комнаты, поэтому пусть они входят в состав объекта-помещения.**

**Для данной задачи существенное значение имеют только два свойства – длина и ширина. Поэтому классы «окна» и «двери» можно объединить в один. Если бы были важны другие свойства (например, толщина стекла, материал двери), то следовало бы для окон создать один класс, а для дверей – другой. Пока обойдемся одним, и все что нам нужно от него – площадь объекта:**

**class WinDoor:**

**def \_\_init\_\_(self, x, y):**

**self.square = x \* y**

**Класс "комната" – это класс-контейнер для окон и дверей. Он должен содержать вызовы класса "окно\_дверь".**

**Хотя помещение не может быть совсем без окон и дверей, но может быть чуланом, дверь которого также оклеивается обоями. Поэтому имеет смысл в конструктор класса вынести только размеры самого помещения, без учета элементов "дизайна", а последние добавлять вызовом специально предназначенного для этого метода, который будет добавлять объекты-компоненты в список.**

**class Room:**

**def \_\_init\_\_(self, x, y, z):**

**self.square = 2 \* z \* (x + y)**

**self.wd = []**

**def add\_wd(self, w, h):**

**self.wd.append(WinDoor(w, h))**

**def work\_surface(self):**

**new\_square = self.square**

**for i in self.wd:**

**new\_square -= i.square**

**return new\_square**

**r1 = Room(6, 3, 2.7)**

**print(r1.square) *# выведет 48.6***

**r1.add\_wd(1, 1)**

**r1.add\_wd(1, 1)**

**r1.add\_wd(1, 2)**

**print(r1.work\_surface()) *# выведет 44.6***

**18 Numpy. Назначение и основные возможности**

**NumPy — это библиотека языка Python, добавляющая поддержку больших многомерных массивов и матриц, вместе с большой библиотекой высокоуровневых (и очень быстрых) математических функций для операций с этими массивами. У этой библиотеки есть несколько важных особенностей, которые сделали ее популярным инструментом. Во-первых, исходный ее код в свободном доступе хранится на** [**GitHub**](https://github.com/numpy/numpy)**, поэтому NumPy называют open-source модулем для** [**Python**](https://blog.skillfactory.ru/komu-i-dlya-chego-nuzhen-python/)**.**

**Во-вторых, библиотека написана на языках C и Fortran. Это компилируемые языки (языки программирования, текст которых преобразуется в машинный код — набор инструкций для конкретного типа процессора. Преобразование происходит с помощью специальной программы-компилятора, благодаря нему вычисления на компилируемых языках происходят быстрее), на которых вычисления производятся гораздо быстрее и эффективнее, чем на интерпретируемых языках (языки программирования, которые не заточены под конкретный тип процессора и могут быть запущены на разных типах устройств). К этим языкам относится и сам** [**Python**](https://blog.skillfactory.ru/komu-i-dlya-chego-nuzhen-python/)**.**

**Где используется NumPy**

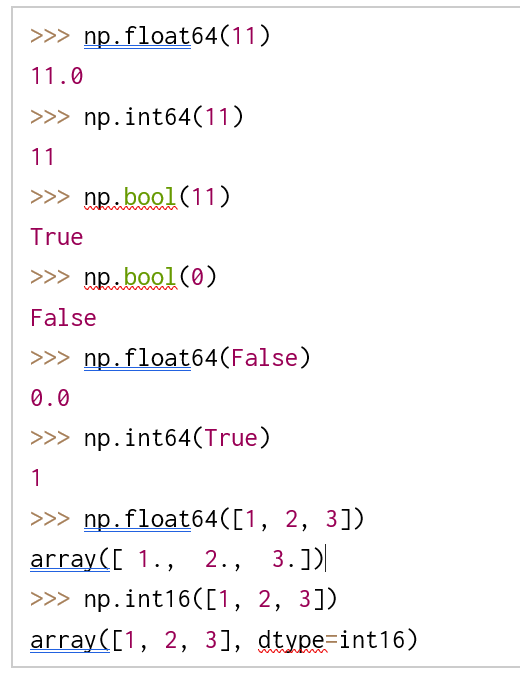
* **Научные вычисления. NumPy пользуются ученые для решения многомерных задач в** [**математике и физике**](https://pyquil-docs.rigetti.com/en/stable/)**,** [**биоинформатике**](https://biopython.org/)**,** [**вычислительной химии**](https://cantera.org/) **и даже** [**когнитивной психологии**](https://www.psychopy.org/)**.**
* **Создание новых массивных библиотек. На основе NumPy появляются новые типы массивов, возможности которых выходят за рамки того, что предлагает библиотека. Например, библиотеки** [**Dask**](https://dask.org/)**,** [**CuPy**](https://cupy.chainer.org/) **или** [**XND**](https://xnd.io/)**.**
* **Data Science. В основе** [**экосистемы**](https://numpy.org/) **для анализа данных лежит NumPy. Библиотека используется на всех этапах работы с данными:** [**извлечение и преобразование**](https://pandas.pydata.org/)**,** [**анализ**](https://jupyter.org/)**,** [**моделирование и оценка**](https://www.statsmodels.org/stable/index.html)**,** [**репрезентация**](https://plotly.com/dash)**.**
* **Machine Learning. Библиотеки для машинного обучения** [**scikit-learn**](https://scikit-learn.org/) **и** [**SciPy**](https://www.scipy.org/) **тоже работают благодаря вычислительным мощностям NumPy.**
* **Визуализация данных. По сравнению непосредственно с** [**Python**](https://blog.skillfactory.ru/komu-i-dlya-chego-nuzhen-python/) **возможности NumPy позволяют исследователям визуализировать наборы данных, которые гораздо больше по размеру. Например, библиотека лежит в основе системы** [**PyViz**](https://pyviz.org/overviews/index.html)**, которая включает в себя десятки программ для визуализации.**

**19 Типы данных в Numpy**

**В NumPy существует 5 базовых числовых типов: булевы числа (тип *bool*: 0 - ложь и 1 - истина), целые числа (тип *int*), беззнаковые целые числа (тип *uint*), вещественные числа (числа с плавающей запятой, тип *float*) и комплексные числа (тип *complex*). У некоторых после указания типа следует количество бит необходимое для хранения такого числа в памяти (16, 32, 64 или 128), но некоторые, такие как *int\_* или *intp* зависят от используемой платформы (32 или 64-разрядные машины).**

**Объекты типа данных являются экземплярами класса numpy.dtype, каждый из которых имеет уникальные характеристики.**

**Для преобразования одного типа данных в другой можно так же воспользоваться объектами типа данных:**

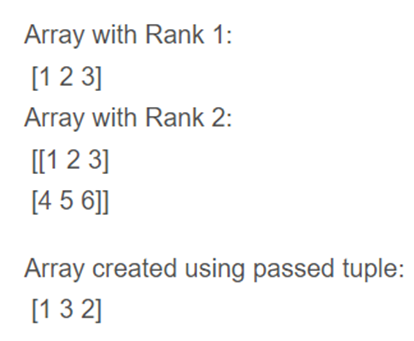
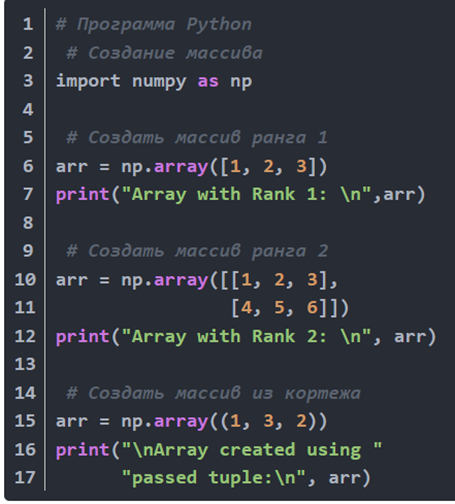
****

**20 Основные операции над массивами и скалярами в Numpy**

**Массив в Numpy - это таблица элементов (обычно чисел), все элементы одного типа, проиндексированные набором натуральных чисел. В Numpy размерность массива называется рангом массива. Кортеж целых чисел дает размер массива в каждом измерении, называемый формой массива. Класс массива в Numpy называется ndarray. Доступ к элементам в массиве Numpy можно получить в квадратных скобках или инициализировать с помощью вложенных списков Python.**

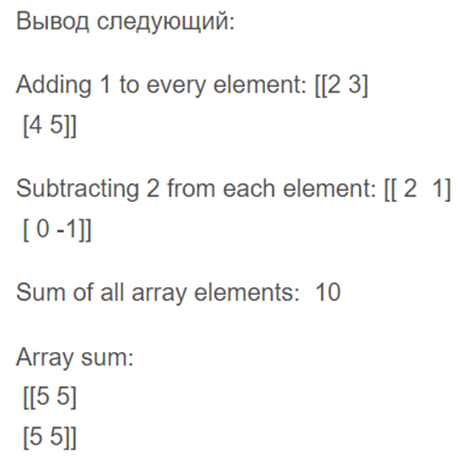
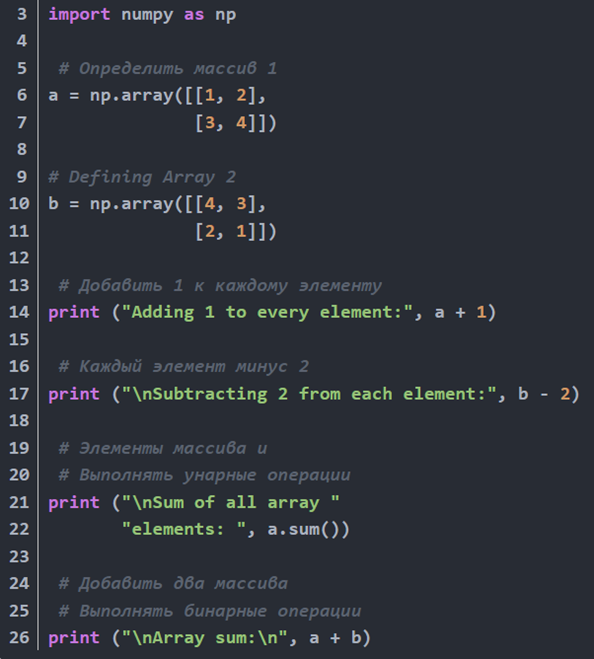
**Массивы в Numpy могут создаваться разными способами, с разным количеством рангов, определяющих размер массива. Массивы также могут быть созданы с использованием различных типов данных (таких как списки, кортежи и т. Д.). Тип составного массива определяется типом элементов в последовательности.**

**Примечание: вы можете явно определить тип массива при создании массива.**

****

**В целом, массивы допускают различные операции, которые можно выполнять с конкретными массивами или комбинациями массивов. Эти операции включают в себя некоторые основные математические операции, а также унарные и двоичные операции.**

**Математические операции над массивами выполняются поэлементно. Создается новый массив, который заполняется результатами действия оператора.**

****

**21 Запись логических условий в виде операций с массивами**

**22 Индексирование и вырезание. Булево индексирование**

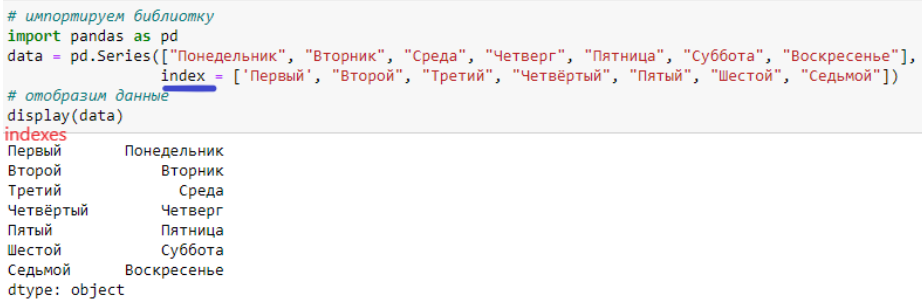
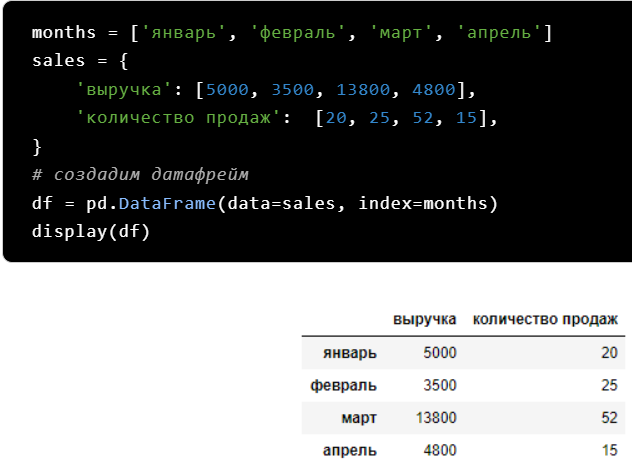
**23 Срезы в Numpy**Срезы позволяют извлекать части массива для создания новых массивов. Когда вы используете [срезы для списков Python](https://pythonru.com/primery/python-spiski-primery), результирующие массивы — это копии, но в NumPy они являются представлениями одного и того же лежащего в основе буфера.  
В зависимости от части массива, которую необходимо извлечь, нужно использовать синтаксис среза; это последовательность числовых значений, разделенная двоеточием (:) в квадратных скобках.  
Чтобы получить, например, часть массива от второго до шестого элемента, необходимо ввести индекс первого элемента — 1 и индекса последнего — 5, разделив их :.  
>>> a = np.arange(10, 16)  
>>> a  
array([10, 11, 12, 13, 14, 15])  
>>> a[1:5]  
array([11, 12, 13, 14])  
Если нужно извлечь элемент из предыдущего отрезка и пропустить один или несколько элементов, можно использовать третье число, которое представляет собой интервал последовательности. Например, со значением 2 результат будет такой.  
>>> a[1:5:2]   
array([11, 13])  
В случае с двухмерными массивами срезы тоже работают, но их нужно определять отдельно для рядов и колонок. Например, если нужно получить только первую строку:  
>>> A = np.arange(10, 19).reshape((3, 3))  
>>> A  
array([[10, 11, 12],

[13, 14, 15],

[16, 17, 18]])  
>>> A[0,:]  
array([10, 11, 12])

**24 Векторизованные операции в Numpy**

**25 Pandas. Назначение и основные возможности**Pandas используется для анализа данных в области науки о данных. Наука о данных — это просто изучение данных с целью получения информации из наборов данных. Набор данных может включать всего несколько записей или миллионы отдельных единиц информации. Задача специалиста по данным — извлечь смысл из этих данных посредством процесса уточнения и анализа. После завершения анализа результаты можно визуализировать с помощью таких инструментов, как Matplotlib, ещё одна библиотека Python.  
1.Внесите изменения в существующий файл. Например, у вас есть электронная таблица Excel. Вы хотите выполнить некоторые вычисления с использованием существующих данных и добавить несколько столбцов, содержащих результаты этих вычислений. С помощью Pandas вы можете импортировать исходную электронную таблицу, произвести вычисления, используя несколько строк кода, а затем сохранить электронную таблицу, чтобы она содержала результаты.  
2.Помогите вам визуализировать данные. После того, как вы очистили свои данные, вы можете визуально представить их с помощью Matplotlib.  
3.Создавайте проекты машинного обучения. Так же, как вы можете объединить Pandas с Matplotlib для удовлетворения своих потребностей в визуализации данных, вы также можете объединить Pandas с Scikit-Learn для выполнения задач машинного обучения.

**26 Типы данных Dataframe и Series в Pandas  
Series** или серии - это одномерные массивы данных. Они похожи на списки, однако, работа с ними отличается. Еще Series можно рассматривать как одну колонку таблицы.  
В стандартном списке мы можем обращаться к элементу по его индексу или порядковому номеру, в Series тоже так можно, но кроме того, в Series индексами послужить данные любого типа: даты, текст, числа, id и др.  
Итак, давайте попробуем создать Series, содержащий названия дней недели и присвоим им текстовые индексы с порядковым номером:  
  
Обычно в работе с данными имеют дело не с единичными значениями, а с целыми **группами данных**, например, в продажах обычно учитывают множество факторов: цену, количество продаж, количество товаров, наличие скидок, средний чек и так далее.  
Для хранения такой информации хорошо подходят **DataFrame** (DataFrame).  
Если Series мы сравнивали с колонкой таблицы, то **DataFrame** - это уже целая **таблица**, у которой есть столбцы (колонки), строки и ячейки.  
Получается, что колонки **DataFrame** - это серии.  
Обычно в одной колонке таблицы хранятся данные, описывающие одну и ту же характеристику. Давайте рассмотрим простой пример, где мы хотим посмотреть на динамику продаж:  
  
Создать **DataFrame** можно двумя способами:  
1.Создать пустой DataFrame и наполнить его данными:  
df = pd.DataFrame()  
2.Создать DataFrame на основе других структур данных стандартного Python, например, из словаря, как мы делали в примере:  
df = pd.DataFrame(data=sales, index=months)  
Метод **.info()** позволяет увидеть информацию о **DataFrame**: названии колонок, количестве элементов, пропусков и типе данных, хранящихся в колонках.

**27 Базовая функциональность Pandas. Доступ по индексу, выборка и фильтрация**

Базовые функции DataFrame

T Переставляет строки и столбцы.

axes Возвращает список с метками оси строк и метками оси столбцов в качестве единственных членов.

dtypes Возвращает типы в этом объекте.

empty Истинно, если NDFrame полностью пуст [нет элементов]; если какая-либо из осей имеет длину 0.

ndim Количество осей / размеров массива.

shape Возвращает кортеж, представляющий размерность DataFrame.

size Количество элементов в NDFrame.

values Numpy представление NDFrame.

head() Возвращает первые n строк.

tail() Возвращает последние n строк.

Выбирать отдельные элементы можно по принципу обычных массивов numpy, используя для этого индекс. Или же можно выбрать метку, соответствующую положению индекса. Таким же образом можно выбрать несколько элементов массива numpy с помощью следующей команды:

>>> s[0:2]

a 12

b -4

dtype: int64

Благодаря тому что основной библиотекой в pandas является NumPy, многие операции, применяемые к массивам NumPy, могут быть использованы и в случае с Series. Одна из таких — фильтрация значений в структуре данных с помощью условий.

Например, если нужно узнать, какие элементы в Series больше 8, то можно написать следующее:

>>> s[s > 8]

a 12

d 9

dtype: int64

**28 Обработка отсутствующих данных в Pandas**

Pandas - это библиотека Python для анализа и обработки данных. Почти все операции в pandas вращаются вокруг DataFrame s, абстрактной структуры данных, специально созданной для обработки метрической тонны данных.

Первым шагом к правильному анализу данных является очистка и организация данных.

Если мы хотим отобразить список значений, которые должны рассматриваться как отсутствующие значения во всех столбцах, мы можем передать список кандидатов, которые мы хотим глобально рассматривать как отсутствующие значения, в параметр na\_values

После того, как мы определили все недостающие значения в DataFrame и правильно аннотировали их, есть несколько способов обработки недостающих данных.

Один из подходов - удалить все строки, содержащие пропущенные значения. Это легко сделать с помощью специально предназначенной для этого функции dropna()

inplace = True вносит все изменения в существующий DataFrame не возвращая новый. Без него вам пришлось бы переназначить DataFrame самому себе.

axis указывает, работаете ли вы со строками или столбцами: 0 - строки, а 1 - столбцы.

Вы можете контролировать , хотите ли вы удалить строки , содержащие по меньшей мере , 1 NaN или все NaN значения, установив , how параметр в dropna методе.

Заполнение недостающих значений:

1) Можно использовать функции .median() , .mode() и .mean() для столбца и указать их в качестве значения заполнения.

2) Вы также можете решить заполнить значения с пометкой NA постоянным значением. Например, вы можете ввести специальную строку или числовое значение.

3) Прямое заполнение отсутствующих значений фрейма данных.

df['Salary'].fillna(method='ffill', inplace=True)

4) Обратное заполнение отсутствующих значений фрейма данных

df['Salary'].fillna(method='bfill', inplace=True)

5) Этот метод использует математическую интерполяцию, чтобы определить, какое значение было бы на месте отсутствующего значения:

df['Salary'].interpolate(method='polynomial')

Очистка и предварительная обработка данных - очень важная часть каждого анализа данных и каждого проекта в области науки о данных.

**29 Статистика в Pandas. Уникальные значения, счётчики значений**

Большое количество методов совместно вычисляет описательную статистику и другие связанные операции над DataFrame. Большинство из них являются агрегатами, такими как sum (), mean (), но некоторые из них, например sumsum () , создают объект одинакового размера. Вообще говоря, эти методы принимают аргумент оси , как и ndarray. {Sum, std, …}, но ось может быть указана по имени или целому числу.

**30 Базовая функциональность Pandas.**

Объекты pandas ( Index , Series , DataFrame ) можно рассматривать как контейнеры для массивов, которые содержат фактические данные и выполняют фактические вычисления. Для многих типов базовым массивом является numpy.ndarray . Однако pandas и сторонние библиотеки могут расширять систему типов NumPy, добавляя поддержку настраиваемых массивов.

Основные функции серии

**axes** Возвращает список меток оси строк

**dtype** Возвращает dtype объекта.

**empty** Возвращает True, если серия пуста.

**ndim** Возвращает количество измерений базовых данных по определению 1.

**size** Возвращает количество элементов в базовых данных.

**values** Возвращает серию как ndarray.

**head()** Возвращает первые n строк.

**tail()** Возвращает последние n строк.

Базовые функции DataFrame

T Переставляет строки и столбцы.

**axes** Возвращает список с метками оси строк и метками оси столбцов в качестве единственных членов.

**dtypes** Возвращает типы в этом объекте.

**empty** Истинно, если NDFrame полностью пуст [нет элементов]; если какая-либо из осей имеет длину 0.

**ndim** Количество осей / размеров массива.

**shape** Возвращает кортеж, представляющий размерность DataFrame.

**size** Количество элементов в NDFrame.

**values** Numpy представление NDFrame.

**head()** Возвращает первые n строк.

**tail()** Возвращает последние n строк.

**31 Переиндексация и удаление элементов в Pandas**

Переиндексирование df.reindex()

После объявления в структуре данных объект Index нельзя менять. Но с помощью операции переиндексирования это можно решить.

>>> ser = pd.Series([2,5,7,4], index=['one','two','three','four']) >>> ser

one 2

two 5

three 7

four 4

dtype: int64

Для того чтобы провести переиндексирование объекта Series библиотека pandas предоставляет функцию reindex(). Она создает новый объект Series со значениями из другого Series, которые теперь переставлены в соответствии с новой последовательностью меток.

При операции переиндексирования можно поменять порядок индексов, удалить некоторые из них или добавить новые. Если метка новая, pandas добавит NaN на место соответствующего значения.

>>> ser.reindex(['three','four','five','one'])

three 7.0

four 4.0

five NaN

one 2.0

dtype: float64

Как видно по выводу, порядок меток можно поменять полностью. Значение, которое раньше соответствовало метке two, удалено, зато есть новое с меткой five.

Удаление

Удалить строку или колонку не составит труда, потому что метки используются для обозначения индексов и названий колонок.

В этом случае pandas предоставляет специальную функцию для этой операции, которая называется drop(). Метод возвращает новый объект без элементов, которые необходимо было удалить.

Например, возьмем в качестве примера случай, где из объекта нужно удалить один элемент. Для этого определим базовый объект Series из четырех элементов с 4 отдельными метками.

>>> ser = pd.Series(np.arange(4.), index=['red','blue','yellow','white'])

>>> ser

red 0.0

blue 1.0

yellow 2.0

white 3.0

dtype: float64

Теперь, предположим, необходимо удалить объект с меткой yellow. Для этого нужно всего лишь указать ее в качестве аргумента функции drop().

>>> ser.drop('yellow')

red 0.0

blue 1.0

white 3.0

dtype: float64

Для удаления большего количества элементов, передайте массив с соответствующими индексами.