Лекция 0 (вводная): Знакомство с ООП на C++.

Вводная лекция строится по типу case study (исследование сразу на примере).

Кто знаком с синтаксисом C-подобных языков, переходит сразу на следующую страницу.

Вдаваться в подробности синтаксиса C++ мы пока не будем. Нам достаточно узнать, как написать простейшую программу на C++, как объявить переменную, хранящую целое число (предположим, что других типов данных и не нужно), как объявить функцию на C++ и как вывести строку и число на консоль.

Простейшая программа на C++ (она ничего не делает ☺):

```
int main()
{
	return 0;
}
```

Объявить целочисленную переменную на C++ можно так:

```
int main()
{
	int age = 20;
	return 0;
}
```

Рассмотрим пример простой функции на C++ (суммы 2 чисел) и вызов этой функции:

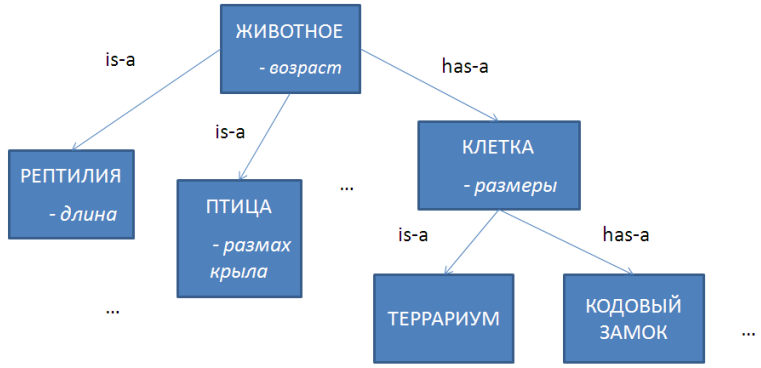
```
// аналог этой записи на паскале - function sum(x,y: integer): integer
int sum(int a, int b)
{
	int result = a + b;
	return result;			// можно сразу – return a + b;
}

int main()
{
	int x1 = 20;
	int x2 = 35;

	int s = sum(x1, x2);			// вызов функции sum;

	std::cout << "Sum is " << s << std::endl;		// выведет: Sum is 55

	return 0;
}
```

Последняя строчка кода – это вывод на консоль строки и переменной s. Для того чтобы она работала, надо в самом начале текста программы подключить специальную библиотеку:

```
#include <iostream>
```

Вот, собственно, все и готово для того, чтобы заняться кейс стади.

Теперь пару слов об ООП. Данная парадигма начала развиваться 40 лет назад, вместе с фреймовой моделью Марвина Минского представления знаний. Рассмотрим эту модель на примере предметной области зоопарка (zoo domain):



Сразу отметим, что современное ООП – это не просто моделирование таксономии (объектов и связей между ними). В рамках ООП программисты кодируют еще абстракции, интерфейсы, контракты, динамику взаимодействия сущностей и многое другое. Но для понимания самых базовых элементов мы начнем, естественно, с тривиальности.

Фрейм, упрощенно говоря, представляет собой выражение некоторой сущности. Фрейм имеет слоты, в которых хранятся данные фрейма, и определенные функции по работе со своими данными. На рисунке слоты выделены курсивом; таких слотов может быть много. Фрейм *как сущность* (например, сущность «рептилия» как абстракция) в терминологии ООП вылился в понятие КЛАССА. Фрейм с заполненными слотами по конкретному экземпляру сущности (например, конкретная рептилия – питон Иннокентий из клетки №18) вылился в понятие ООП «ОБЪЕКТ КЛАССА».

Основные связи между фреймами – это «имеет» (has-a) и «является (уточнением)» (is-a). Смысл этих связей из картинки интуитивно понятен: например, у животного ЕСТЬ клетка, а птица – это ЧАСТНЫЙ СЛУЧАЙ животного. Первый тип связи в ООП реализуется в виде **агрегации** или **композиции** объектов, второй тип – в виде **наследования** классов. Это все центральные понятия ООП.

Теперь просто рассмотрим, как можно закодировать на C++ часть приведенной выше модели.

Для этого создадим в MS Visual Studio новый проект типа Win32 Console Application. При создании в окне свойств поставим чекбокс «Empty project» (пустой проект). В окне Solution Explorer («обозреватель решений») увидим три пустые папки – Header Files, Resource Files и Source Files (их смысл рассмотрим на следующих лекциях). Добавим к папке «Source Files» файл main.cpp (создадим его – правой кнопкой Add New… и в категории CPP files введем main.cpp).

И напишем немного кода, помимо функции int main(). Мы добавим класс «Животное» с одним членом (свойством) – *возраст*, потому что это та характеристика, которая присуща любому животному, начиная от амебы и заканчивая соседом, который орет за стенкой в три часа ночи ☺. Также предположим, что животное может повзрослеть на год. Эта функция оформится в виде метода void growOlder() класса:

```
// ============================================================ класс животное, живое существо
class Animal
{
public:                         // публичные (открытые) данные класса:
        int age_;               // у животного есть возраст (целое число)

        void growOlder()        // и предположим, что любое животное мы можем заставить повзрослеть
        {
                age_++;         // это C++-стайл аналог записи age = age + 1
        }
};
// ============================================================================================

int main()                              // а в главной функции...
{
        Animal a;                       // создаем животное “a” (объект класса)
        a.age_ = 15;                            // присвоим ему возраст 15 (можно, т.к. все данные открыты)
        cout << a.age_ << endl;         // выведет на экран «15»

        a.growOlder();                          // пусть повзрослеет
        std::cout << a.age_ << std::endl; // выведет на экран «16»

        return 0;
}
```

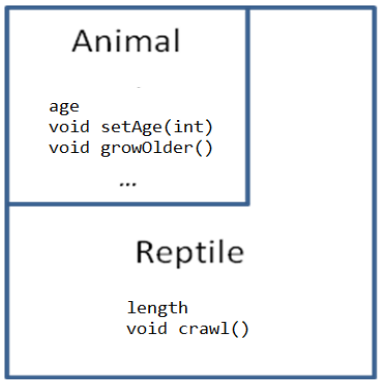
Первый принцип ООП – **ИНКАПСУЛЯЦИЯ**.
В широком смысле означает, что внутреннее содержимое и подробности работы одного модуля не должны быть известны другим модулям, и какие-либо внутренние изменения в этом модуле не должны отражаться на других. *В узком смысле* заключается в том, что данные классов скрываются внутри самих классов, т.е. доступ к ним должен быть возможен только из самих классов (или из наследников). Для указания этого факта применяется **спецификатор доступа** private (или protected) вместо public. Но при этом становится вопрос – как работать с закрытыми данными? Во-первых, для работы с ними обычно создаются так называемые ГЕТТЕРЫ (получатели значений свойств) и СЕТТЕРЫ (установщики значений свойств). Во-вторых, начальные значения свойств задаются в КОНСТРУКТОРАХ классов – это специальные методы класса, которые неявно вызываются при создании класса. Таким образом, код нужно переписать так:

```
class Animal
{
public:
        Animal()                // конструктор; при создании животного просто обнулим возраст
        {
                age_ = 0;
        }

        void setAge(int age)            // Сеттер: установить значение возраста
        {
                age_ = age;
        }

        int getAge()                            // Геттер: узнать возраст
```

```
	{
		return age_;
	}

	void growOlder()
	{
		age_++;
	}

private:
	int age_;		// скрыли от посторонних доступ к возрасту
};


int main()
{
	Animal a;		// здесь неявно вызывается конструктор: age_ = 0
	a.age_ = 15;			// компилятор ругается – возраст то закрыт от посторонних!
	a.setAge(15);			// а вот так можно! Задать возраст через спец.метод - сеттер

	std::cout << a.getAge() << std::endl;		// и узнать возраст можно! Через геттер

	return 0;
}
```

Обратите внимание, что теперь нельзя написать a.age_ – компилятор выдаст ошибку, т.к. данное свойство класса теперь закрыто.
Второй принцип ООП – **НАСЛЕДОВАНИЕ**.
Наследование – это не что иное, как способ закодировать отношение типа is-a. При наследовании класс-потомок имеет доступ ко всем данным класса-родителя (наследует их), кроме тех, которые объявлены private, следовательно, и у рептилии есть метод для взросления на 1 год. У рептилии был бы и возраст, но в прошлом примере мы присвоили ему уровень доступа private. Впрочем, сеттеры и геттеры возраста достались в наследство рептилии. Можно также открыть доступ к возрасту и для рептилии, при этом оставляя его закрытым для всех остальных – это делается через спецификатор protected (об этом также в следующих лекциях). В памяти объект класса-потомка строится «поверх» объекта класса-родителя.
Грамотно отнаследовать Рептилию от Животного можно, например, так:

```
class Reptile: public Animal    // наследование через :
{
public:
	Reptile()
	{
		length_ = 0;
	}

	void crawl()
	{
		length_++;
	}
    ...
private:
	int length_;  // данные конкретно для рептилии
	  // + родительские неявно достались
};
```



Для рассмотрения третьего главного принципа ООП – **ПОЛИМОРФИЗМА** – нам пока не хватает знаний, касающихся работы с указателями и динамической памятью в C++, поэтому оставим его на потом. Но суть его можно рассмотреть уже сейчас. Смысл полиморфизма заключается в том, что функционал (вызываемый метод) будет определяться во время выполнения программы, а не на этапе компиляции. Например, предположим, что рептилия имеет свою «версию взросления» и взрослеет за раз не на 1 год, а на целых 2. Полиморфизм позволяет динамически определять, метод какого класса нужно будет вызвать в каждом конкретном случае. Если мы создадим целый массив животных, часть из которых будут рептилиями, и пройдемся по этому массиву, вызывая метод growOlder, то для рептилий будет вызываться своя версия метода, а для остальных животных – своя, хотя вызывать мы будем одну и ту же функцию в коде.

Наконец, **АГРЕГАЦИЯ/КОМПОЗИЦИЯ** позволяет реализовать связь типа has-a. Например, как связать животное и клетку? Ответ довольно прост и интуитивно понятен: включить ссылку (более точно - *указатель*) на класс «Клетка» в класс «Животное» (либо непосредственно сам объект класса «Клетка», а не ссылку; в данном случае это будет называться композиция, в первом случае – агрегация (более общий тип связи, при котором часть и целое могут существовать во времени независимо друг от друга)).
Объявим класс «Клетка» и создадим объект Клетки в Животном:

```
class Cage
{
public:
	void setWidth(int width)
	{
		width_ = width;
	}

	int getWidth()
	{
```

```
            return width_;
        }
        ...
private:
        int width_;
        int height_;
};


class Animal
{
public:
        Animal()
        {
            age_ = 0;
        }
        ...
private:
        int age_;
        Cage cage_;                    // Композиция: в животном сразу и информация о клетке для него
};
```

Отметим также, что в этой лекции мы о многом не сказали и кое-что упростили. В частности, объявлять разные классы в одном cpp-файле – дурной тон. По ходу всего курса мы эти пробелы закроем и неточности исправим.

Лекция 1. Работа с памятью.

- Указатели и адреса
- Ссылки
- Указатели и константность
- Передача параметров в функцию
- Классы памяти
- Одномерные массивы
- Арифметика указателей
- Двумерные массивы
- Строки
- Указатель void*
- Указатель на функцию

```
int main()
        {
            // ************************************
            // ===== PART 1. УКАЗАТЕЛИ И АДРЕСА ======
            // ************************************
            int value = 45;
            double coeff = -113.29;
            // операция & - получить адрес переменной в памяти
            std::cout << &value << std::endl;   // 00 22 f7 98      (адрес value, этот адрес может, естественно, отличаться в каждом
        конкретном случае)
            std::cout << &coeff << std::endl;   // 00 22 f7 90   (адрес переменной coeff в памяти)
            int* valuePtr = &value;             // адрес можно присвоить переменной типа "указатель" (int*): 00 22 f7 98
            double* coeffPtr = &coeff;
            // операция * - разыменовать указатель (получить те данные в памяти, на которые он указывает)
            std::cout << *valuePtr << std::endl;

            // ************************************
            // ========== PART 2. ССЫЛКИ ============
            // ************************************
            int& ref = value;         // алиас, псевдоним для переменной value
                                // в данном случае & означает "ссылку" (другое имя для переменной value)
            ref++;                  // "нарастили" ссылку
            std::cout << value << std::endl;    // а в итоге нарастилась также и та переменная, на которую ссылается ref
            value++;
            std::cout << ref << std::endl;      // ref и value ссылаются на одни и те же данные

            // ************************************
            // ====== PART 3. УКАЗАТЕЛИ и Const ======
            // ************************************
            int ten = 10, twenty = 20;
            const int* ptr1 = &ten;   // указатель на неизменяемое значение
            //(*ptr1)++;              // так сделать нельзя
            ptr1 = &value;            // а сам указатель изменить можно
```

```
    int* const ptr2 = &twenty;          // неизменямый (!) указатель (сам)
    (*ptr2)++;                          // изменить то, на что он указывает, можно, а сам указатель - нельзя
    //ptr2 = &value;            // так сделать нельзя
    std::cout << twenty << std::endl;
    const int* const ptr3 = &ten;
    std::cout << *ptr3 << std::endl;

    // ************************************
    // ====== PART 4. ПАРАМЕТРЫ ФУНКЦИЙ ======
    // ************************************
    int x1 = 15;
    int x2 = 40;
    int res = sum(x1, x2);    // передаем параметры по ссылке (см. сигнатуру функции sum()),
                              // поэтому x1 изменится в теле функции (т.к. там меняется ее ссылка - a).
                              // А вообще параметры функций копируются в локальные переменные функций
                              // и существуют только в теле функции
    std::cout << "x1 = " << x1 << std::endl;
    std::cout << "Sum = " << res << std::endl;

    // ************************************
    // ======= PART 5. КЛАССЫ ПАМЯТИ =========
    // ************************************
    // code         - memory where code instructions are stored
    // globals      - memory for all static variables and globals
    // STACK        - "fast" memory
    // HEAP         - "big" memory

    int inner = 90; // это создано и живет в стеке
    // специально создадим блок {...} для демонстрации области действия переменных (scope)
    {
        int* innerPtr = &inner;             // указатель на inner
        (*innerPtr)++;
        int* heapPtr = new int;             // оператор new выделяет память в куче (HEAP) - в данном случае там выделилось 4
байта под int
        *heapPtr = 798;                     // и в эту ячейку записали значение 798
    }
                      // здесь оба указателя (хранившиеся в стеке) уже уничтожатся (по 4 байта),
                      // т.к. выйдут из области действия {...}
    inner++;                  // с inner будет все в порядке, т.к. уничтожился только указатель на нее
    std::cout << inner << std::endl;
    // а кусок памяти, в котором хранится 798 будет бесполезно занимать место в куче,
    // т.к. единственный указатель на него уничтожен
    // это утечка памяти - memory leak

    int* val = new int;       // опять выделить память в HEAP под один int
    *val = 35;                // присвоить по указателю данные 35
    delete val;               // освободить в памяти HEAP (теперь утечки памяти не будет, когда в конце main() указатель val
будет уничтожен)
    val = NULL;     // delete освобождает память, но ничего не делает с указателем. Здесь сами делаем val = 0;
    // Кстати, здесь есть важнейший нюанс!
    // - мы можем спокойно удалить нулевой указатель:
    //   val = NULL;
    //   delete val;
    // Спонсор нашего спокойствия - стандарт C++, согласно которому компилятор просто проигнорирует delete.
    // Но (!) вызов delete два раза подряд для одного и того же ненулевого указателя
    // приводит к самым нежелательным буквам в мире C++ - UB (Undefined Behaviour),
    // а конкретно в этом случае почти наверняка crash программы. Т.е. так делать нужно забыть:
    //     delete x;
    //     delete x;

    // ************************************
    // ======== PART 6. МАССИВЫ (1D) =========
    // ************************************
    // пример массива, объявленного статически (живет в стеке) и с инициализацией сразу
    int arr[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    std::cout << arr[0] << std::endl;   // возможен доступ по индексу - как обычно
    int* heapArray = new int [5];       // пример массива, объявленного динамически (живет в куче, а в стеке живет указатель
на первый элемент)
```

```
    for (size_t i = 0; i < 5; ++i)      // пример другой инициализации - в цикле числами 42, 43, 44, 45, 46
    {                                   // NB. Тип std::size_t - в данном случае эквивалент unsigned int
        heapArray[i] = 42 + i;
    }
    std::cout << heapArray << std::endl;    // оказывается, массив в C++ - это суть не что иное, как указатель на первый
элемент
    std::cout << *heapArray << std::endl;   // так мы получим первый элемент (разыменовав переменную, которая "отвечает" за
массив)
    // так можно узнать размер массива на стеке
    std::cout << sizeof(arr) / sizeof(arr[0]) << std::endl;
    // передаем массив в функцию (передача параметра по указателю - поэтому массив будет меняться)
    processArray(arr, 10);

    // ************************************
    // ==== PART 7. АРИФМЕТИКА УКАЗАТЕЛЕЙ ====
    // ************************************
    int* pointer = arr;
    pointer++;                // нарастили указатель на 1.
                        // это не означает, что мы увеличили адрес ячейки памяти на 1.
                        // А означает, что мы продвинулись вперед на то количество байт, которое занимает тип при указателе
(int*)
                        // Это т.н. арифметика указателей
    std::cout << *heapArray << std::endl;         // т.е. здесь будет второй элемент массива
                                                 // (хотя указатель мы сдвинули на 1, а не на 4 (байта))
    short* shortArray = (short*)heapArray;        // а вот если попытаемся "шагать" по массиву 2-байтовыми шагами (short*) ...
    for (size_t i = 0; i < 5; ++i)
    {
        *(shortArray + i) = 42 + i;     // (эта запись эквивалентна shortArray[i]
                                        // (оператор [] был введен в C++ просто для большей читабельности))
    }
    std::cout << *heapArray << std::endl;   // ... то здесь уже будет не то число, т.к. мы в 4-байтовой ячейке оказались два
раза
                                            // в предыдущем цикле (шагая по 2 байта)
    delete []heapArray;
    // ************************************
    // ======== PART 8. МАССИВЫ (2D) =========
    // ************************************
    //int arr2D[2][6];                          // пример двумерного массива 2х6, объявленного статически

    int** arr2D = new int* [2];                 // пример двумерного массива 2х6, объявленного динамически
                                        // это по сути указатель на указатели (или массив массивов)
                                        // здесь мы создаем массив указателей (массив строк)

    for (size_t i = 0; i < 2; ++i)      // а здесь мы создаем массив по каждой строке массива
    {
        arr2D[i] = new int [6];
    }
    // ... делаем что-нибудь с массивом ...
    // например, присвоим последнему элементу значение 73
    arr2D[1][5] = 73;
    for (size_t i = 0; i < 2; ++i)  // так мы освобождаем память, выделенную под двумерный массив
    {
        delete [] arr2D[i];
    }
    delete [] arr2D;
    // ************************************
    // ========== PART 9. СТРОКИ ============
    // ************************************
    char text[100] = { 'H', 'e', 'l', 'l', 'o' };
    wchar_t widetext[100] = L"Hello!";
    std::cout << text << std::endl;
    std::wcout << widetext << std::endl;
    // Есть C-style функции для работы со строками, например, strcat():
    char newText[100] = " world!";
    strcat(text, newText);
    std::cout << text << std::endl;
    // В C++ есть удобный и стандартный строковый тип std::string
```

```
    std::string str = "Hello";
    std::wstring wstr = L"Hello";
    str.append(" world!");
    std::cout << str << std::endl;
    wstr.append(L" world!");
    std::wcout << wstr << std::endl;

    // ***************************************
    // ====== PART 10. УКАЗАТЕЛЬ VOID* =======
    // ***************************************
    // Прототип функции memset
    // std::memset(void* destination, int value, size_t num)
    int numbers[128];
    std::memset(numbers, 0, 128*sizeof(int));
    // Прототип функции memcpy
    // std::memcpy(void* destination, const void* src, size_t num)
    int coeffs[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
    std::memcpy(coeffs, numbers, 5*sizeof(int));
    printArray(coeffs, 10);
    // приведенные функции работают не только с массивами, а вообще
    // с любыми данными в памяти, т.к. в памяти это все наборы байт

    // ***************************************
    // ==== PART 11. УКАЗАТЕЛИ НА ФУНКЦИЮ ====
    // ***************************************
    void (*arrayFunc)(int[], size_t) = processArray;
    arrayFunc(arr, 10);
    arrayFunc = printArray;
    arrayFunc(arr, 10);
    // указатель arrayFunc имеет свойства указателей, а значит:
    // - его можно перебросить на другую функцию (с аналогичным прототипом)
    // - его можно передать в другую функцию в параметре
    // - можно сделать массив таких указателей и "прогонять" массив данных по всем функциям из него
    return 0;
}
```

Лекция 2. Классы и объекты, инкапсуляция.

- Классы Car и Driver, создание машин и водителей на стеке и в куче
- Структура ООП-проекта, разделение интерфейсной части класса и реализационной (.h и .cpp)
- Header guards: два способа
- Конструкторы: по умолчанию, параметризированные, explicit, списки инициализации
- Копирующий конструктор, поверхностное (shallow) и глубокое (deep) копирование
- Инкапсуляция в узком смысле, спецификаторы public и private, сеттеры/геттеры
- Константные объекты классов, const и mutable
- Вызов метода класса с точки зрения компилятора, указатель this
- Статические члены и методы класса
- Запрет создания объекта класса

Part 1 - Car

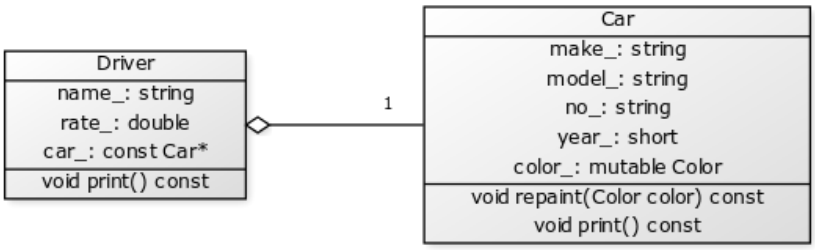
На примере класса машины демонстрируется простая инкапсуляция, перегрузка конструкторов, конструктор Car(short year) без спецификатора explicit, списки инициализации в конструкторах, делегирование конструкторов в c++11, создание объектов на стеке и в хипе. Также некоторые Car-объекты (ford и lexus) сделаны константными. Чистые функции (pure functions) и неизменяемые объекты (immutable objects) - это хорошо по многим причинам. Показано, зачем нужно ключевое слово mutable на примере члена color_, если "константную"" машину захотят перекрасить.

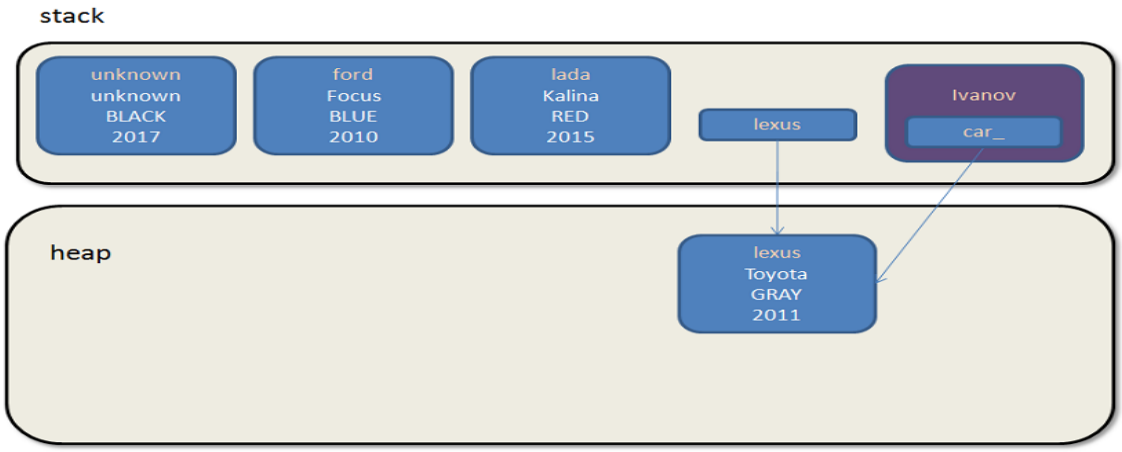
Part 2 - Driver

На примере класса водителя демонстрируется концепция глубокого и поверхностного копирования объектов класса, статические члены класса на примере (по сути глобального) топ-рейтинга водителей top_rate (у каждого водителя есть свой рейтинг rate_, а максимальный всегда содержится в статической переменной в качестве ориентира для всех водитилей), какими способами к ним можно обращаться, а также что делает компилятор при вызове метода класса на объекте и откуда берется указатель this.

Part 3 - Misc

В файле misc.h объявлены классы Empty, Quiz, GlobalScreen для демонстрации некоторых нюансов классов в языке C++: размер объекта пустого класса; методы класса, создаваемые компилятором C++ по умолчанию; бессмысленность вызова конструктора в теле конструктора; inline-методы класса; запрет создания объекта класса путем помещения его конструктора в секцию private.






```
int main() {
                    // ========================== PART 1 ==============================
                    // Класс Car: демо конструкторов, сеттеров, const, mutable
                    // ================================================================
                    // 1) создаем ОБЪЕКТ unknown КЛАССА Car на стеке : вызовется конструктор без параметров
                   Car unknown;
                   // 2) создаем *константный* ОБЪЕКТ ford КЛАССА Car на стеке :
                   //    вызовется основной параметризированный конструктор
                   const Car ford("Ford", "Focus", "e888ap", Color::BLUE, 2010);
                   // 3) создаем ОБЪЕКТ lada КЛАССА Car на стеке : вызовется конструктор с одним параметром (!)
                   Car lada = 2015;
                   // полностью эквивалентно такой записи (см. также explicit конструктор в классе Driver):
                   // Car lada(2015);
                   lada.setMake("LADA");
                   lada.setModel("Kalina");
                   lada.setNo("o379ac");
                   lada.setColor(Color::RED);
                   // так не получится, потому что данные - private:
                   // lada.make_ = "LADA"
                   // lada.model_ = "Kalina"
                   // ...
                   // 4) создаем константный ОБЪЕКТ lexus КЛАССА Car в хипе :
                   //    вызовется основной параметризированный конструктор
                   const Car* lexus = new Car("Toyota", "Lexus", "a777aa", Color::GRAY, 2011);
                   // хоть lexus и константный объект,
                   // но его color_ объявлен как mutable, а метод repaint() - с модификатором const,
                   // поэтому можно вызвать этот метод:
                   lexus->repaint(Color::WHITE);
                   std::cout << std::endl;
                   unknown.print();
                   ford.print();
                   lada.print();
                   lexus->print();
                   std::cout << std::endl;
                   // еще можно так написать: на ходу создать объект класса и вызвать его метод
                   Car().print();                                          // 1
                   std::cout << std::endl;
                   Car("VAZ", "2109", "c111cc", Color::BLACK, 2007).print();   // 2
                   std::cout << std::endl;
```

```
    // ========================== PART 2 ================================
    // Класс Driver: демо explicit конструктора, статических членов и this
    // ==================================================================
    Driver ivanov("Ivanov", lada);
    ivanov.print();
    std::cout << "Two years later..." << std::endl;
    ivanov.setCar(*lexus);
    ivanov.print();
    Driver petrov("Petrov");  // создать объект можно только явным вызовом конструктора, т.к. он помечен
explicit!
                              // вот так уже не сойдет (как в случае с Car lada = 2015):
                              // Driver petrov = std::string("Petrov");
    petrov.print();
    ivanov.setRate(7.9);
    petrov.setRate(8.4);
    // Обращение к статическим членам может идти по-разному:
    std::cout << "Top rate among drivers: " << std::endl;
    std::cout << Driver::top_rate << std::endl;
    std::cout << ivanov.top_rate << std::endl;
    std::cout << petrov.top_rate << std::endl;
    // но суть одинакова: статические данные -
    // это по сути глобальные данные, помещенные в scope класса

    // ========================== PART 3 ================================
    // Классы из файла misc.h: демо пустого класса и приватного конструктора
    // ==================================================================
    std::cout << std::endl << "misc demos:" << std::endl;
    Empty empty;
    std::cout << "Size of empty: " << sizeof(empty) << std::endl;
    // Закомментированный код не скомпилируется, т.к. конструктор - private:
    // GlobalScreen screen;
    // screen.hello();
    GlobalScreen::hello();
    GlobalScreen::bye();
    std::cout << std::endl;
    // напоследок полюбуемся лексусом Иванова )))
    const Car* car = ivanov.getCar();
    car->print();
    delete lexus;
    // если попробуем снова написать так:
    //
    // car->print();
    //
    // то вылетим, т.к. car Иванова указывает на lexus, который мы только что удалили.
    // Может смутить такая кажущаяся ненадежность кода,
    // но в нашем случае мы концептуально так и решили изначально -
    // возложить ответственность за сохранность данных не на водителей,
    // а на внешний код - здесь это функция main(), она должна следить за ссылками
    return 0;
}


class Empty
{
};
// 1) ради интереса - сколько байт занимает объект пустого класса?
//    Ответ: 1 байт
// 2) он таки пустой-пустой?
//    Ответ: нет, для любого класса компилятор создает автоматически 4 метода
//           (если программист не написал их сам):
/*
class Empty
```

```
{
public:
    // 1) конструктор по умолчанию
    Empty();
    // 2) копирующий конструктор
    Empty(const Empty& empty);
    // 3) оператор присваивания
    Empty& operator=(const Empty& empty);
    // 4) деструктор
    ~Empty();
    // в c++11 создаются еще 2 метода с move-семантикой:
    // 5)
    Empty(Empty&& empty);
    // 6)
    Empty& operator=(Empty&& empty);
};
*/
```

Лекция 3. Перегрузка операторов.

- Общий смысл перегрузки операторов
- Перегрузка операторов в виде методов класса и в виде глобальных функций
- Особенности возврата из функции копии, указателя и ссылки, RVO
- Перегрузка бинарных операторов на примере "+" и "<"
- Перегрузка оператора индексации "[]", обсуждение l-value
- Перегрузка префиксного и постфиксного инкремента/декремента
- Перегрузка оператора присваивания, идиома copy-and-swap
- Rule of Big Three

Класс заказа Booking

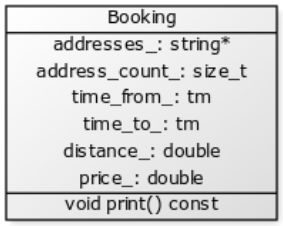
В информацию о клиентском заказе у нас будет пока входить:

- массив адресов заказа *(в большинстве случаев адрес отправки и назначения, но могут быть и промежуточные пункты)*
- время отправки клиента
- время доставки клиента
- расстояние
- стоимость

Вместо тысячи слов продемонстрируем пример использования класса с перегруженными операторами:

```
Booking booking_artema("Artema Str., 516", "Artema Str., 100", time1);
Booking booking_circus("Circus", "Planetarium", time2);


// можно добавить еще адрес к заказу через операцию +=

booking_circus += "Donbass Arena";


booking_artema.print();
booking_circus.print();


// заказ можно инкрементировать;
// смысл этой операции для заказа - перенести время заказа на 1 минуту

Booking copy = ++booking_circus;

std::cout << "Booking Circus -> ... has been altered to 1 minute later" << std::endl;

booking_circus.print();
copy.print();            // должно вывести то же самое



// пункты-адреса заказа можно получать по индексу

booking_circus[0] = booking_artema[1];

std::cout << "Change address in booking Circus -> ..." << std::endl;
booking_circus.print();


// скомбинировать два заказа
```

```
    // (теперь заказ охватывает все адреса из заказа1 и заказа2)

    Booking booking = booking_artema + booking_circus;
    booking.print();


    // сравнить два заказа:
    // если заказ был сделан раньше, то он "меньше"

    if (booking_artema < booking_circus)
    {
        std::cout << "Booking Artema preceeds booking Circus" << std::endl;
    }
    else
    {
        std::cout << "Booking Circus preceeds booking Artema" << std::endl;
    }
```




```
// Конструктор без параметров
                                // все равно выделяет память под два неизвестных пока адреса
                                // и устанавливает время в текущее
                                Booking::Booking() : distance_(0.0), price_(0.0)
                                {
                                    addresses_ = new std::string[2] { "unknown", "unknown" };       // C++11 initialization
                                    address_count_ = 2;
                                    std::time_t now = std::time(0);
                                    time_from_ = *localtime(&now);
                                }
                                // Конструктор с двумя адресами и временем оформления заказа
                                Booking::Booking(const std::string& address_from,
                                                 const std::string& address_to,
                                                 tm time)
                                    : time_from_(time), time_to_(time), distance_(0.0), price_(0.0)
                                {
                                    addresses_ = new std::string[2] { address_from, address_to };
                                    address_count_ = 2;
                                }
                                // Конструктор с массивом адресов и временем оформления заказа
                                Booking::Booking(const std::string addresses[],
                                                 std::size_t address_count,
                                                 tm time)
                                    : time_from_(time), time_to_(time), distance_(0.0), price_(0.0)
                                {
                                    addresses_ = new std::string [address_count];
                                    for (std::size_t i = 0; i < address_count; ++i)
                                    {
                                        addresses_[i] = addresses[i];
                                    }
                                    address_count_ = address_count;
                                }
                                // Т.к. объекты Booking держат динамические данные,
                                // следующие два метода надо обязательно переопределить:
                                // Копирующий конструктор ...
                                Booking::Booking(const Booking &b)
                                {
                                    // глубокое копирование массива адресов:
                                    address_count_ = b.address_count_;
                                    addresses_ = new std::string [address_count_];
                                    for (std::size_t i = 0; i < address_count_; ++i)
                                    {
                                        addresses_[i] = b.addresses_[i];
                                    }
                                    // копируем все остальное:
```

```
    time_from_ = b.time_from_;
    time_to_ = b.time_to_;
    distance_ = b.distance_;
    price_ = b.price_;
}
// ... и оператор присваивания тут как тут
Booking& Booking::operator=(Booking b)
{
    swap(b);        // делаем модно - идиома copy-and-swap
    return *this;
}
// copy-and-swap idiom
void Booking::swap(Booking b)
{
    std::swap(addresses_, b.addresses_);
    std::swap(address_count_, b.address_count_);
    std::swap(time_from_, b.time_from_);
    std::swap(time_to_, b.time_to_);
    std::swap(distance_, b.distance_);
    std::swap(price_, b.price_);
}
/*
// обычный способ перегрузки оператора присваивания
// (похож на конструктор копирования, только надо удалить прежний массив адресов)
Booking& Booking::operator=(const Booking& b)
{
    // обязательно проверяем это:
    if (this == &b)
        return *this;
    delete []addresses_;
    // копируем всё
    address_count_ = b.address_count_;
    addresses_ = new std::string [address_count_];
    for (std::size_t i = 0; i < address_count_; ++i)
    {
        addresses_[i] = b.addresses_[i];
    }
    time_from_ = b.time_from_;
    time_to_ = b.time_to_;
    distance_ = b.distance_;
    price_ = b.price_;
    return *this;
}
*/
// скажем утечкам памяти "нет"
Booking::~Booking()
{
    delete []addresses_;
}
std::string* Booking::getAddresses() const
{
    return addresses_;
}
std::size_t Booking::getAddressCount() const
{
    return address_count_;
}
tm Booking::getTime() const
{
    return time_from_;
}
void Booking::print() const
{
    std::cout << "Booking "
              << time_from_.tm_hour << ":" << time_from_.tm_min << ":" << time_from_.tm_sec
              << std::endl;
    for (std::size_t i = 0; i < address_count_ - 1; ++i)
    {
```

```
        std::cout << addresses_[i] << " -> ";
    }
    std::cout << addresses_[address_count_ - 1];
    std::cout << std::endl << std::endl;
}
// ====================== ПЕРЕГРУЗКА ОПЕРАТОРОВ ======================
// по-хорошему здесь надо проверить выход index за "пределы разумного"
// (но пока не делаем эту проверку)
std::string& Booking::operator[](std::size_t index) const
{
    return addresses_[index];
}
// префиксный инкремент
Booking& Booking::operator++()
{
    time_from_.tm_min++;
    return *this;
}
// постфиксный инкремент
Booking Booking::operator++(int)
{
    Booking b(*this);
    time_from_.tm_min++;
    return b;
}
Booking& Booking::operator+=(const std::string& address)
{
    std::string* addresses = new std::string [address_count_ + 1];
    for (std::size_t i = 0; i < address_count_; ++i)
    {
        addresses[i] = addresses_[i];
    }
    addresses[address_count_++] = address;
    delete []addresses_;
    addresses_ = addresses;
    return *this;
}
// ============================================================
// ========== Внимание! Это глобальные функции ============
// ==== Если их сделать friend, то кода будет поменьше ====
// ============= Но это тема другой лекции ================
// ============================================================
// простая логика
bool operator<(const Booking& b1, const Booking& b2)
{
    tm time1 = b1.getTime();
    tm time2 = b2.getTime();
    return difftime(mktime(&time1), mktime(&time2)) < 0;
}
// чуть более хитрая логика
Booking operator+(const Booking& b1, const Booking& b2)
{
    std::size_t count1 = b1.getAddressCount();
    std::size_t count2 = b2.getAddressCount();
    std::size_t total_count = count1 + count2;
    std::string* addresses1 = b1.getAddresses();
    std::string* addresses2 = b2.getAddresses();
    // дополнительная плюшка - проверяем, а не является ли
    // конечный пункт первого заказа начальным пунктом второго
    if (addresses1[count1 - 1] == addresses2[0])
    {
        // если да, то не будем дублировать последний пункт первого заказа
        total_count--;
        count1--;
    }
    std::string* addresses = new std::string [total_count];
    // конкатениируем два массива адресов:
    std::size_t i = 0, j = 0;
```

```
    while (i < count1)
    {
        addresses[i++] = addresses1[j++];
    }
    j = 0;
    while (i < total_count)
    {
        addresses[i++] = addresses2[j++];
    }
    // и создаем комбинированный заказ:
    Booking b(addresses, total_count, b1.getTime());
    delete []addresses;
    return b;
}
// =====================================================
```

Лекция 4. Наследование.

- Наследование как принцип ООП и механизм повторного использования кода, отношение **is-a**
- Спецификатор доступа protected, явный доступ к данным базового класса
- Конструкторы и деструкторы подклассов, порядок создания и удаления классов и подклассов
- Доопределение базовых классов, классы и подклассы в памяти
- Переопределение членов-данных и методов базовых классов
- Сокрытие, инструкция using
- Антипаттерн "Отказ от наследства"
- Виды наследования по доступу: закрытое, защищенное, открытое *(C++ only)*
- Множественное наследование, виртуальное наследование *(C++ only)*
- Заметки кулхацкера: хоть прайвэт, хоть протэктэд - доступ прямо из мейна

Классы Car и Truck - наследники класса Vehicle

До этого у нас был класс Car, которым мы кодировали сущность легкового автомобиля. Но наш такси-сервис катает не только людей, а и их чемоданы ("Гена, а давай я возьму чемодан, а ты возьмешь меня?" (c) Чебурашка). Поэтому нормальным решением будет выделить класс автомобиля Vehicle и отнаследовать от него легковой автомобиль (Car) и грузовой (Truck). Концептуально они будут пока отличаться лишь тем, что у грузового есть свойство максимально возможной загрузки (max_weight_), у легкового выделено отдельное свойство - количество мест (seat_count_).
С технической точки зрения на примере класса Truck демонстрируется доопределение и переопределение метода void print() const базового класса Vehicle, а также затаившееся-бажное переопределение базового свойства color_ в подклассе.

Классы Operator и Driver - наследники класса Employee

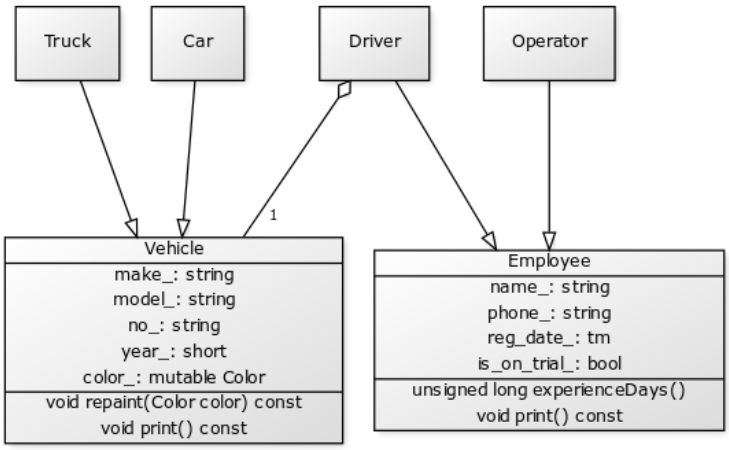
На примере класса Operator демонстрируется решение вопроса с сокрытием метода void print() const базового класса Employee. Подробнее - см. в коде.

Хэдер misc.h

Здесь собран код для демонстрации вопросов, указанных в последних 3 пунктах плана лекции.
В конце функции int main() обходим всякие там private и protected:

```
// ================ Эй хакнем! ================

EasyToHack lolkek;
lolkek.print();

// lolkek.foo   = 300;        // не получится
// lolkek.bar_ = 6.28;     // не получится

// а так:
int* iptr = (int*)&lolkek;

*iptr = 300;        // !

iptr++;             // смещаемся дальше на размер инта
iptr++;             // вопрос на засыпку: а еще раз зачем??

double* dptr = (double*)iptr;
*dptr = 6.28;

lolkek.print();
```

Классы на данный момент:



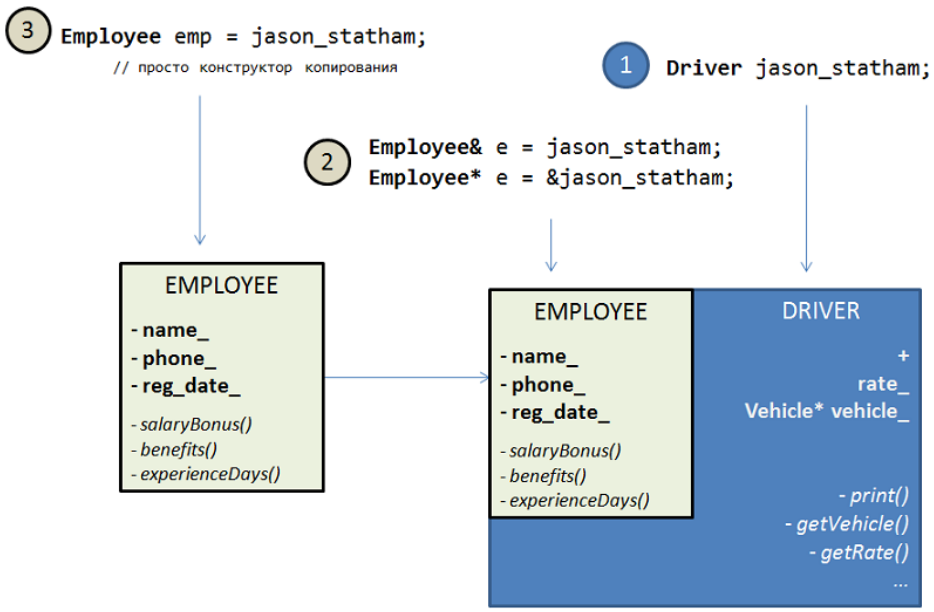

```
int main()
            {
                // демонстрация подкласса на примере класса Car
                {
                    Car car("Toyota", "Lexus", "a965ea", Color::WHITE, 2013);
                    // ================== последовательность вызова конструкторов:
                    // -> car.Vehicle();
                    // -> car.Car();
                    // ================= деструкторы вызываются в обратном порядке
                    car.print();      // print() достался от базового Vehicle
                }
                std::cout << std::endl;
                // демонстрация доопределения, переопределения с помощью класса Truck
                {
                    Truck truck;
                    // у Truck есть свои методы в дополнение к методам родителя
                    // Например, сеттер маскимального веса. Это *доопределение*
                    truck.setMaxWeight(1500);
                    // Truck определяет свою версию print()
                    // это *переопределение* (не путать с сокрытием)
                    truck.print();
                    // а так мы можем доступиться до родительского print()
                    // (равно как и до любых публичных данных родителя)
                    truck.Vehicle::print();
                    // демо проблемного участка
                    // (переопределение Vehicle::color_ в подклассе Truck)
                    // truck.repaint() затронет родительский (Vehicle) color_!
                    truck.repaint(Color::GREEN);
                    if (truck.getColor() == Color::GREEN)
                    {
                        std::cout << "The truck is green now!" << std:: endl;
                    }
                    else
                    {
                        std::cout << "The truck is NOT green!" << std:: endl;
                    }
                    if (truck.Vehicle::getColor() == Color::GREEN)
                    {
                        std::cout << "The truck's parent is green now!" << std:: endl;
                    }
                    else
                    {
                        std::cout << "The truck's parent is NOT green!" << std:: endl;
                    }
                }
                std::cout << std::endl;
                // демонстрация сокрытия с помощью классов Operator и Employee
                {
                    std::time_t now = std::time(0);
                    tm time = *localtime(&now);
                    Operator sidorova("Sidorova A.E.", "000-111-22-33", time);
                    sidorova.setId(17614);
```

```
            // Т.к. есть using, то print() сотрудника "просочился" в неймспейс оператора
            sidorova.print(true);
            std::cout << std::endl << "Print as employee:" << std::endl;
            sidorova.print();
            // без using внутри Operator тут будет ошибка
        }
        std::cout << std::endl;
        // демонстрация результата private-наследования
        PrivatelyInherited demo;
        // есть доступ только к методу print()
        demo.print();
        std::cout << std::endl;
        return 0;
    }
```

Лекция 5. Полиморфизм, интерфейсы.

- Полиморфизм в широком смысле:
  - перегрузка функций
  - перегрузка операторов
  - полиморфизм подтипа
  - шаблоны
- Полиморфизм как принцип ООП (полиморфизм подтипа)
- Раннее и позднее связывание, таблицы виртуальных функций
- Виртуальный деструктор
- Виртуальный конструктор (такая штука вообще есть?)
- Полиморфизм и закрытое наследование (*C++ only*)
- Абстрактные классы, чисто виртуальные методы
- Пример вложенного (nested-) класса
- Интерфейсы: технические и философские аспекты
- Последний пункт можно вполне считать центральным в объектно-ориентированном проектировании и программировании.

Полиморфизм на примере классов Employee, Operator, Driver
Сначала проведен технический осмотр механизма полиморфизма. Подробности - см. в функции main().

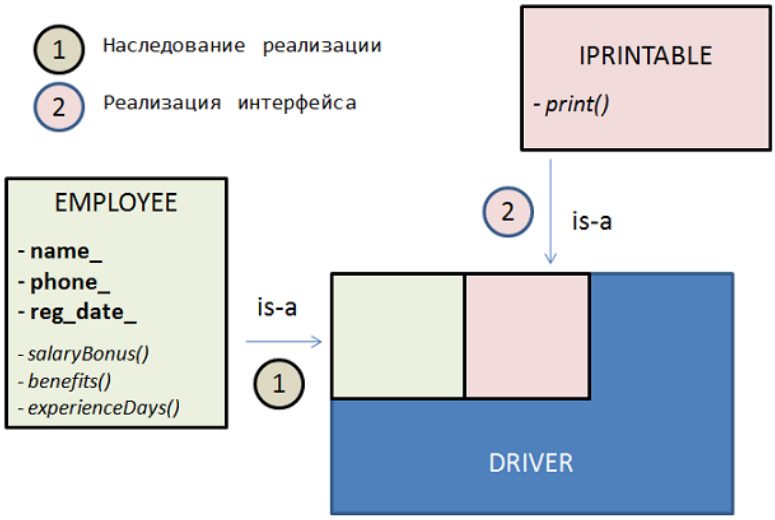


Абстрактный класс GeoService и его реализация TaxiGeoService
Далее в проект был введен класс гео-службы GeoService.
В нем есть как конкретика (координаты центра города, методы для расчета расстояний между объектами и преобразования адресов в геокоординаты и обратно), так и сугубо абстрактная "вещь" - составление маршрута от адреса к адресу, а также оценка времени доставки клиента к месту назначения (это только идея для гео-службы, идея, которую выражают уже конкретные реализации гео-службы).
В случае с абстракциями можно говорить, что класс уже не просто наследуется от другого класса (наследование реализации), а реализует его "интерфейс" (наследование интерфейса, реализация интерфейса).

Класс **Гео-служба такси** не просто *является* гео-службой (is-a), а еще конкретным образом реализует *идею*составления маршрута от адреса к адресу, а также оценки времени доставки клиента к месту назначения.

Интерфейс IPrintable

Если абстрактный класс вообще не имеет никакой конкретики, а служит только как выражение намерений программиста, то он называется **интерфейсом** (к сожалению, в C++ не выделено отдельное ключевое слово для такой важной концепции, кроме miscrosoft-specific __interface). В нашем проекте выделим пока один простейший интерфейс IPrintable - этот интерфейс задает функционал отображения объекта.



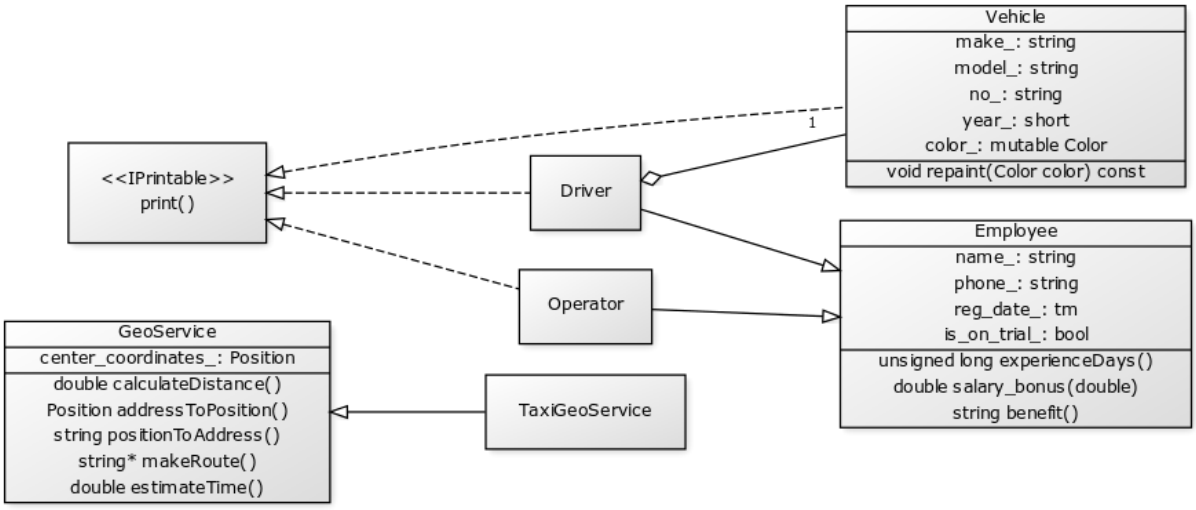
Интерфейс - это просто абстрактный класс, доведенный до фанатизма?

Интерфейс - это только технически класс, а концептуально - это способ программисту закодировать не сущность, а некоторую идею и свои намерения. Если угодно, по интерфейсам в проекте может (по крайней мере, должно) стать понятно, как эта система задумывалась работать хотя бы на уровне взаимодействия отдельных объектов.

Пишут ли люди вообще пустые интерфейсы?

Да. Хотя технически это ничего не дает (кроме оверхеда в 4/8 байт на размер объектов всех классов, реализующих данный пустой интерфейс), такие интерфейсы могут значительно облегчить понимание тех или иных конкретных классов (один из элементов самодокументирования кода).

Все классы, игравшие в этой лекции:




```
int main()
            {
                // ======================== PART 1 ========================
                // ==================== ПОЛИМОРФИЗМ ======================
                // ========================================================
                Employee emp("Somebody 2 luv", "000-888-77-22", tm());
                Operator ivanova("Ivanova A.E.", "000-555-88-99", tm());
                // здесь уже на этапе компиляции понятно,
                // метод какого класса будет вызываться:
                std::cout << "Employee's bonus: " << emp.salaryBonus(1000) << std::endl;
                std::cout << "Operator's bonus: " << ivanova.salaryBonus(1000) << std::endl;
                // а теперь морально готовимся к полиморфизму:
```

```
Employee* employee = new Employee("Somebody 2 luv", "000-111-22-23", tm());
Employee* petrova = new Operator("Petrova A.E.", "000-111-22-33", tm());
// Внимательно! Тут написано что-то новенькое:
//
//      Employee* X = new Operator Y;
//
// Т.е. тип слева и справа от оператора присваивания разный
// Но так написать нельзя:
//
// Operator* person = new Employee;
//
// Чтобы понять почему так, просто вспомним, как в памяти хранится
// родитель и подкласс
// ... Прэподаватель рисует на доске напоминание ...
// Благодаря полиморфизму, выведутся разные надбавки к зарплате
// (здесь результат такой же, как и выше, но по совершенно другой причине!
//  здесь только во время ВЫПОЛНЕНИЯ программы бло решено, какой метод вызвать)
std::cout << "Employee's bonus: " << employee->salaryBonus(1000) << std::endl;
std::cout << "Operator's bonus: " <<  petrova->salaryBonus(1000) << std::endl;
// Однакож, если метод не будет помечен как virtual, то полиморфизма не будет!
// и будет вызываться метод класса Employee в обоих случаях.
// Специально для демо мы сделали benefit() не виртуальным, а обычным
std::cout << "Employee's benefits: " << employee->benefits() << std::endl;
std::cout << "Operator's benefits: " <<  petrova->benefits() << std::endl;
// Если что, в Java слово virtual не нужно,
// потому что ВСЕ методы класса там являются виртуальными по умолчанию
// Еще раз приведем в порядок мысли:
// 1) virtual делает метод полиморфным,
//    но если метод вызывается на стековых объектах, полиморфизма не будет
//    (будет работать механизм простого переопределения метода)
// 2) если метод не помечен как virtual,
//    то вызывающий класс будет определен еще на этапе компиляции
//    на основании типа, стоящего ДО оператора присваивания
// стандартная для универов по всему миру демонстрация полиморфизма
// на примере массива из указателей:
Employee* employees[4] =
{
    new Employee("Vasechka"),
    new Operator("Johnny"),
    new Driver("Eddie"),
    &ivanova
};
std::cout << "Bonuses for 1) Vasechka, 2) Johnny, 3) Eddie, 4) Ivanova" << std::endl;
for (std::size_t i = 0; i < 4; ++i)
{
    std::cout << "Bonus " << i+1 << ": " << employees[i]->salaryBonus(1000) << std::endl;
}
// привент зе мемори лик
for (std::size_t i = 0; i < 4; ++i)
{
    delete employees[i];
}
// Полиморфизм будет происходить и с ссылками (&) :
// Driver driver;
// Employee& employee = driver;
// std::cout << employee.salaryBonus(1000) << std::endl;
// а теперь убедимся, что таблицы виртуальных функций таки существуют:
class A
{
public:
    void foo();
};
class ASubclass : public A {};
class VirtualA
{
public:
    virtual void foo() {}
};
```

```
    // любой подкласс также получает доп. указатель
    class VirtualASubclass : public VirtualA {};
    std::cout << "sizeof(A) = " << sizeof(A) << std::endl;
    std::cout << "sizeof(ASubclass) = " << sizeof(ASubclass) << std::endl;
    std::cout << "sizeof(VirtualA) = " << sizeof(VirtualA) << std::endl;
    std::cout << "sizeof(VirtualASubclass) = " << sizeof(VirtualASubclass) << std::endl;
    // ====================== PART 2 =========================
    // ========== Демонстрация абстрактного класса =============
    // ==========================================================
    std::cout << std::endl << "GeoService abstract class demo" << std::endl;
    TaxiGeoService geo_service({41.523, 23.872});
    TaxiGeoService::Position position =
            geo_service.addressToPosition("Donetsk, Artema, 500");
    std::cout << position.langitude << " " << position.latitude << std::endl;
    // так не получится (и правильно):
    // 1) GeoService gs;                        // ни так
    // 2) GeoService* gs = new GeoService;      // ни так
    // так нормально:
    GeoService* gs = new TaxiGeoService({41.523, 23.872});
    std::string* addresses = gs->makeRoute("Donetsk, DonNU", "Donetsk, DonNTU");
    std::cout << addresses[0] << " -> " << addresses[1] << std::endl;
    delete []addresses;     // довольно неприятный момент -
                            // вытекание ресурса из метода наружу
                            // (лечится разными способами, об этом - не сейчас)
    delete gs;
    // и так нормально:
    GeoService& service = geo_service;
    std::cout << service.calculateDistance("Donetsk, DonNU", "Donetsk, DonNTU") << " km" << std::endl;
    //
    // Т.е. создать можно только какую-то реализацию абстракции (ее конкретное наполнение)
    // ================================================================================
    // ====================== PART 3 =========================
    // ========== Демонстрация интерфейса IPrintable ===========
    // ==========================================================
    std::cout << std::endl << "IPrintable interface demo" << std::endl;
    Driver* vasya = new Driver("Vasya");
    Vehicle car("Ford", "Focus", "x915ae", Color::WHITE, 2012);
    showObjectAddress(ivanova);
    showObjectAddress(*vasya);
    showObjectAddress(car);
    // а эти не скомпилируются, потому что мы не обозначили, что
    // в классах Employee и TaxiGeoService есть функционал вывода на экран
    // (они не реализуют интерфейс IPrintable)
    // show_object_address(*employee);
    // show_object_address(*petrova);
    // show_object_address(geo_service);
    // Подумайте, а чем petrova то провинилась? ))
    delete employee;
    delete petrova;
    delete vasya;
    // Бонус: нюанс с закрытым наследованием
    // Предположим, у нас есть такие классы:
    /*
        class Parent
        {
            // ...
        };
        class Child: private Parent
        {
             // ...
        };
        В клиентском коде
        компилятор даже не даст написать так:
        Parent* c = new Child;
        // подумайте, почему
     */
    return 0;
}
```

```
// вот эта функция, которая принимает любой принтэбл объект
void showObjectAddress(IPrintable& printable)
{
    std::cout << " === The object === " << std::endl;
    printable.print();
    std::cout << " === has address === " << std::endl;
    std::cout << &printable << std::endl << std::endl;
}
    // ================================================ In general:
    // Polymorphism:
    //
    // 1. Subclass polymorphism
    // 2. Function overload:    print()   print(std::string header)
    //                                                                  =>          =>
    //                          print()   print("aaa");
    // 3. int x = 2 + 3
    //    Point p = p1 + p2;
    // 4. C++ Templates
    // ============================================================
/*
 *
 * еще пример интерфейса:
 *
 *
enum FileFormat
{
    TXT,
    XML,
    JSON
};
// Интерфейс конвертирования файлов одного формата в другой
class IFileConverter
{
    virtual void convert(FileFormat inFormat, FileFormat outFormat) = 0;
    virtual void save() = 0;
    virtual ~IFileConverter() {}
};
*/
```

Лекция 6. Динамические преобразования типа, RTTI.

- RTTI (Run-Time Type Identification) в C++
- Преобразования типов в C++:
  - static_cast<>
  - dynamic_cast<>
  - const_cast<>
  - reinterpret_cast<>
- Некоторые замечания по использованию dynamic_cast<>

Вопросы демонстрируются на примере классов Employee, Operator и Driver.
Код довольно подробно прокомментирован (см. функцию int main()).

```
int main()
        {
            // Механизмов т.н. рефлексии C++ "из коробки" не дает,
            // но какую-никакую интроспекцию осуществить можно
            // 2 способа:
            // 1) оператор typeid, структура type_info
            // 2) динамический каст dynamic_cast<>
            // Поработаем еще с массивчиком из прошлой лекции:
            Operator ivanova("Ivanova A.E.");
            const Vehicle car("Ford", "Focus", "x276ae", Color::WHITE, 2011);
            Employee* employees[4] =
            {
                new Employee("Vasechka"),
                new Operator("Johnny"),
                new Driver("Eddie", car),
                &ivanova
            };
```

```
    // Можно ли динамически узнать, на объект какого типа в хипе указывает каждый указатель?
    // Воспользуемся typeid...
    // но такой код ничего нам не дает:
    for (std::size_t i = 0; i < 4; ++i)
    {
        std::cout << typeid(employees[i]).name() << std::endl;
        std::cout << typeid(employees[i]).__is_pointer_p() << std::endl;
    }
    // а такой - вуаля!
    for (std::size_t i = 0; i < 4; ++i)
    {
        std::cout << typeid(*employees[i]).name() << std::endl;
        std::cout << typeid(*employees[i]).__is_pointer_p() << std::endl;
    }
    for (std::size_t i = 0; i < 4; ++i)
    {
        Driver* driver = dynamic_cast<Driver*>(employees[i]);
        if (driver)
        {
            std::cout << driver->getName() << " drives " << driver->getVehicle()->getModel() <<
std::endl;
        }
        else
        {
            std::cout << employees[i]->getName() << " is not a driver" << std::endl;
        }
    }
    // "А нужна ли вообще аналитика?" (с) Галыгин и Харламов
    // а нужен ли вообще dynamic_cast<>, если можно сделать так в сишном стиле:
    Driver* vodila = (Driver*)employees[2];         // тут таки водитель
    std::cout << vodila->getName() << std::endl;
    // Да, здесь нам повезло.
    // Впрочем, первая строчка никогда не таит опасности,
    // а вот - вторая может подкинуть проблем
    vodila = (Driver*)employees[1];                 // тут оператор
    std::cout << vodila << std::endl;
    // выводит какой-то адрес, но не ноль!
    // т.е. каст "типа произошел", но доступ по указателю к данным именно водительской части
    // вышвырнет нас куда подальше
    // std::cout << vodila->getVehicle()->getModel() << std::endl;
    // Указатель может быть 0/nullptr, а если работаем с ссылками?
    Driver jason_statham("Jason Statham");
    Employee& employee1 = jason_statham;
    Operator penny("Penny");
    Employee& employee2 = penny;
    // здесь все нормально
    Driver& d = dynamic_cast<Driver&>(employee1);
    std::cout << d.getName() << std::endl;
    // а здесь никакие проверки не помогут,
    // т.к. каст бросит исключение bad_cast;
    // исключение надо перехватывать (об исключениях - через лекцию)
    // d = dynamic_cast<Driver&>(employee2);
    // ======================== Семейство cast-ов в C++ ======================
    // static_cast<> - делает преобразование еще на этапе компиляции
    //
    // это самый безопасный каст из всех,
    // т.к. ошибка не позволит даже скомпилироваться программе
    int pi = static_cast<int>(3.1415);
    std::cout << pi << std::endl;
    // сишный статический каст более опасен:
    int hun = 100;
    // тут ошибки не всплывет (и плохо)
    double* ptr = (double*)&hun;
    // здесь вылетим, потому что указатель проброшен на 4-байтовую область
    // а сам указатель - на double, т.е. будет попытка записать 8 байт на место четырех
    // *ptr = 45.56;
    // а статик_каст не скомпилируется здесь (и хорошо):
    // int *ptr = static_cast<int*>(&e);
```

```
    // для иерархий классов-подклассов статик_каст использовать не надо!
    // в данном случае все будет нормально,
    // но и статик_каст на другой подкласс тоже пройдет успешно
    // и вылетим уже потом при доступе
    // ======================================================== не надо так
    vodila = static_cast<Driver*>(employees[2]);
    std::cout << vodila->getVehicle()->getModel() << std::endl;
    // ======================================================== не надо так
    // демонстрация const_cast
    // возможное применение - снять константность/волатильность с указателя/ссылки,
    // если какая-то внешняя 3rd-party функция прнимает не константный параметр
    our_function("Hello world!");
    // демонстрация reinterpret_cast
    int chars = 0x61626364;
    // трактуем int как последовательность 4 char-ов
    char* str = reinterpret_cast<char*>(&chars);
    for (std::size_t i = 0; i < 4; ++i)
    {
        std::cout << str[i] << std::endl;
    }
    // little-endian
    // подчистим память
    for (std::size_t i = 0; i < 4; ++i)
    {
        delete employees[i];
    }
    return 0;
}
// чтобы вызвать эту функцию, нужно подать ссылку не константную строку
int library_function(std::string& s)
{
    return s.length();
}
// а у нас, допустим, она константная
// значит, надо явно снять константность
void our_function(const std::string& s)
{
    std::string& tmp = const_cast<std::string&>(s);
    int n = library_function(tmp);
    std::cout << n << std::endl;
}
```

Лекция 7. Агрегаци/композиция, наследование, дружественность.

- Связи типа *has-a*, агрегация, композиция
- Дружественные классы, методы, функции
- Дружественность и перегрузка операторов
- Копаясь в <iostream>...
- Агрегация/композиция vs. Наследование

Итак, у нас уже седьмая лекция, и только сейчас мы закодим хоть какую-то динамику в нашем проекте)). Нельзя сказать, что это напрямую связано с агрегацией/композицией; просто пришло время... Для пущей философскости последней фразы завершу ее словами "подумай над этим..."

Класс-фасад TaxiService

Итак, к этому моменту у нас уже есть пример агрегации: водитель **Driver** агрегирует машину **Vehicle**. Соберем весь ворох классов, нажитых в предыдущих лекциях непосильным трудом, под одной крышей - классом **TaxiService**. Но это не просто технический шаг, ведь когда мы думаем о службе такси, то и представляем, что там есть какие-то операторы и водители с машинами, фиксируются и обрабатываются заказы и т.д.; значит, так и надо проектировать (возможно, потом потребуется рефакторинг, но первоначальный дизайн чаще всего делается на основе таких простых интуитивных представлений). Этот класс будет скрывать в своих недрах, как именно он там работает со всеми своими объектами, а клиентскому коду будет предоставлять некоторый набор удобных функций для управления своим состоянием, которое в том числе складывается из состояний объектов (собственно, инкапсуляция, как она и есть). Такие классы называются *фасадами*.
Класс **TaxiService** реализует более строгую форму агрегации и является *композитом* для всех операторов, водителей и машин службы такси, т.е. в классе хранятся массивы этих объектов, которые будут созданы при создании службы такси и удалены, когда ее удалим. Массивы - это временно, скоро на арену выйдет STL.
Итак, где же та самая динамика? Здеся:

// основная функция - обработка звонка клиента

```
void TaxiService::processCall(std::tm call_time)
{
   // сначала поручаем оператору сформировать заказ:

   Booking* booking = operator->processBooking(call_time);

   // затем поручаем нашему сервису подобрать водителя под этот заказ

   Driver* driver = findDriver(*booking);

   // покажем клиенту информацию по заказу:

   std::cout << "Your car: ";
   driver->getVehicle()->print();

   // здесь поручим гео-службе оценить расстояние и время заказа
   // и выведем клиенту тоже

   std::cout << "Estimated distance and time: "
          << geo_service->calculateDistance(
               (*booking)[0],
               (*booking)[1]) << " km; "
          << geo_service->estimateTime(
               (*booking)[0],
               (*booking)[1]) << " seconds"
          << std::endl;


   // проэмулируем отработку заказа водителем:

   driver->handleBooking(*booking);

   // добавим заказ в журнал заказов:

   bookings  [bookings  count  ++] = booking;
}
```

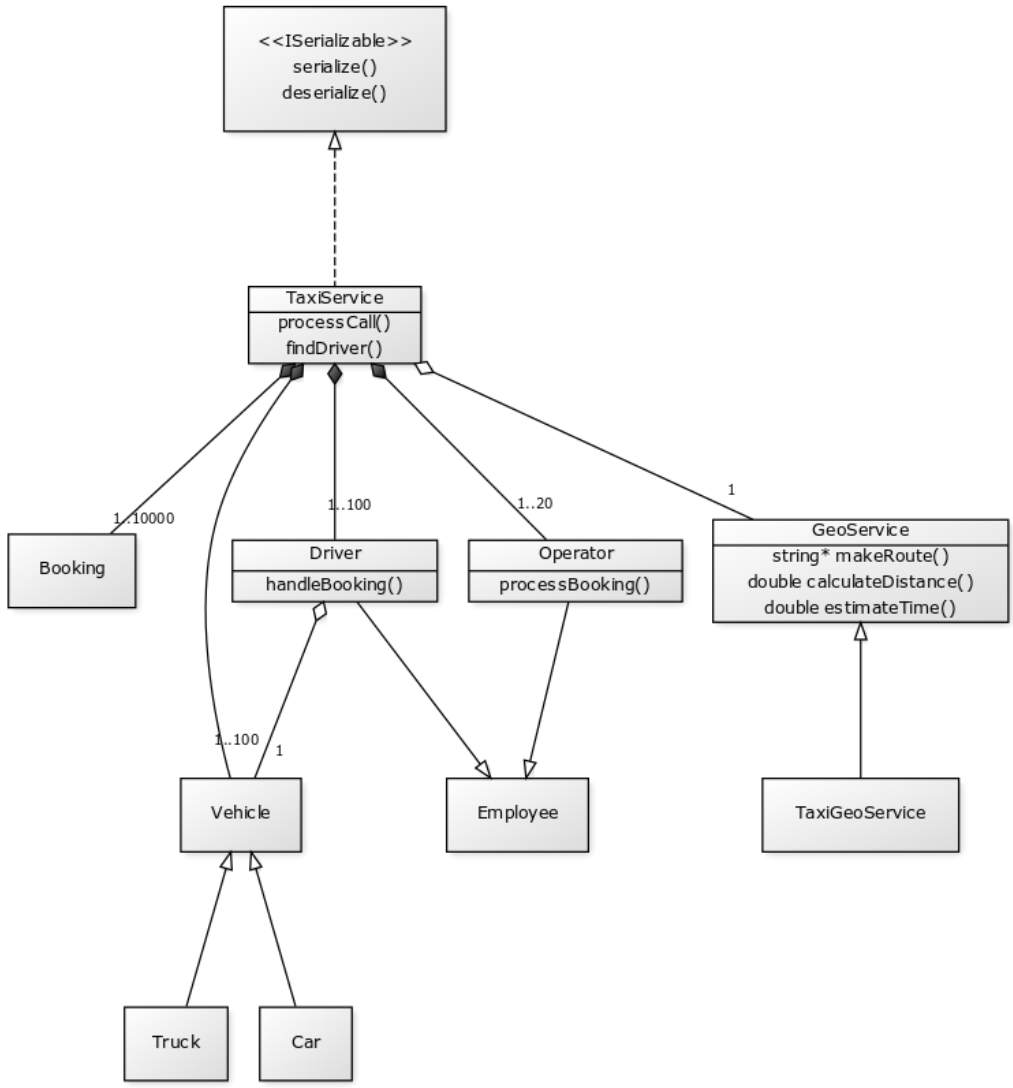
Код не требует детального разжевывания. Отметим, сколько классов вовлечено в действо: тут тебе и заказ Booking, и Operator, который формирует первоначальную версию этого самого заказа (на основе вопросов "Откуда ехать будете?", "Куда Вам?" и "Легковой или грузовой?"), и сама служба такси, которая подыскивает автоматически водителя (findDriver), и водитель Driver, который заказ выполняет, и даже дергаются методы гео-службы GeoService. А с точки зрения клиента всего лишь вызывается метод processCall() с датой-временем звонка клиента.
В соответствии с темой лекции, для демонстрации дружественности также изменен класс Booking - теперь перегруженные операторы стали friend + написан оператор<< для работы с потоками вывода. А еще реализована самая правильная версия метода swap() для реализации идиомы copy-and-swap (еще более правильная, чем была в прошлой версии).
В функции main() проэмулирована обработка двух звонков, после чего данные записываются в файл taxi.txt (такая простенькая текстовая сериализация, для этого добавили в проект еще интерфейс **ISerializable**).
Обработка одного звонка в консоли выглядит примерно следующим образом:
Файл taxi.txt будет выглядеть как-то так:

```
Booking 8:15:20 -> 8:25:20
Digital Planetarium -> Donbass Arena

Booking 9:44:32 -> 9:54:32
Teatralniy, 13 -> Universitetskaya, 24
```

```
void TaxiService::processCall(std::tm call_time)
{
    // сначала поручаем оператору сформировать заказ:
    Booking* booking = operators_[0]->processBooking(call_time);
    // затем поручаем нашему сервису подобрать водителя под этот заказ
    Driver* driver = findDriver(*booking);
    // покажем клиенту информацию по заказу:
    std::cout << "Your car: ";
    driver->getVehicle()->print();
    // здесь поручим гео-службе оценить расстояние и время заказа
    // и выведем клиенту тоже
    std::cout << "Estimated distance and time: "
              << geo_service->calculateDistance(
                     (*booking)[0],
```

```
                    (*booking)[1]) << " km; "
              << geo_service->estimateTime(
                    (*booking)[0],
                    (*booking)[1]) << " seconds"
              << std::endl;
    // проэмулируем отработку заказа водителем:
    driver->handleBooking(*booking);
    // добавим заказ в журнал заказов:
    bookings_[bookings_count_++] = booking;
}
// Функция для поиска водителя
// по основным 2 критериям:
// 1) нужен легковой или грузовой автомобиль
// 2) заказ VIP или нет
Driver* TaxiService::findDriver(const Booking& booking) const
{
    // для VIP-заказа рейтинг водителя должен быть не меньше 0.8
    double min_rate = booking.isVip() ? 0.8 : 0.0;
    for (std::size_t i = 0; i < 100; ++i)
    {
        if (drivers_[i]->hasTruck() == booking.isTruck() &&
            drivers_[i]->getRate() >= min_rate)
        {
            return drivers_[i];
        }
    }
}
```

Последний заказ был сложным - аж 10 минут на грузовике ехали)).
Ну и посмотрим на UML-картинку на данный момент (интерфейс IPrintable не нарисован, чтобы не зашумлять диаграмму):



Лекция 8. Исключения.

- Исключения как способ передачи управления, альтернативы исключениям
- Синтаксис throw, try, catch
- Раскручивание стека (stack unwinding)
- Исключение в конструкторе
- Исключение в деструкторе
- Собственный класс исключения, std::exception
- Еще стандартные исключения (std::out_of_range, std::bad_cast и др.)
- SEH (структурированная обработка исключений)
- Цена исключений
- Разговор о safe-code (инварианты, assert, RAII, TOCTOU и др.)

Лекция начинается с довольно "технической" темы, а заканчивается кодерскими размышлениями "ваапше о жизни".
В основном, сопровождающий код сосредоточен в функции main(). В проекте TaxiService изменения коснулись класса **Booking** - теперь в его перегруженном индексаторе генерируется стандартное исключение std::out_of_range при выходе за границы массива адресов маршрута такси. Также добавлен для примера класс пользовательского исключения **WrongAddressException**, имя которого отвечает само за себя. Этот класс почти идентичен стандартному std::invalid_argument, но еще содержит дополнительное описательное поле типа стек-трейса.

```
int main()
            {
                // ====================================== Стратегии обработки ошибок
                // 1) Return Codes:
                // например, есть функция, которая может возвращать разные коды ошибок
                // openFile() ->
                //    const int FILE_SUCCESS = 0    - успешное открытие файла
                //    const int FILE_NOT_FOUND = 1  - файл не найден
                //    const int FILE_IS BUSY = 2    - файл занят
```

```
//
// int errno = openFile();
// if (errno == FILE_NOT_FOUND)
// {
//     // ...
//     return;
// }
// else if (errno == FILE_IS_BUSY)
// {
//     // ...
//     return;
// }
// при этом, куски кода, которые подчищают память и делают
// другую подчищающую работу, могут дублироваться (а это не DRY)
// Также неудобно, если вызывались вложенные функции
// и нужно при каком-то флаге во внутренней функции
// сразу вернуться в самую "верхнюю" функцию
// 2) Exceptions
// демонстрация #1 работы с exception-ами
// (механизм и синтаксис try..throw..catch)
try
{
    exception_demo();
}
catch (const int error)
{
    std::cout << "Error " << error << std::endl;
}
catch (const char* message)
{
    std::cout << message << std::endl;
}
catch (...)
{
    std::cout << "Something mysterious happened!" << std::endl;
}
// ПРИМЕЧАНИЕ.
// catch(int) и catch(double) - разные вещи! инт не поднимется до дабла автоматически!
// демонстрация #2 работы с exception-ами
// (исключения в более ООП-шном ключе)
std::time_t now = std::time(0);
Booking booking("address1", "address2", *localtime(&now));
try
{
    booking[0] = booking[5];
    std::cout << booking;
}
catch (std::out_of_range& exc)
{
    std::cout << exc.what() << std::endl;
}
Employee petrov("Petrov A.E.");
try
{
    Operator& person = dynamic_cast<Operator&>(petrov);
    person.print();
    person.processBooking(tm());
}
catch (const std::bad_cast& ex)
{
    std::cout << "Some problem!" << std::endl;
    std::cout << ex.what() << std::endl;
}
// а если не знаем или забыли, какое исключение
// генерируется в этом случае? тогда можно перехватить
// базовый std::exception по ссылке и заюзать RTTI:
try
{
```

```
        Operator& person = dynamic_cast<Operator&>(petrov);
        person.print();
        person.processBooking(tm());
    }
    catch (const std::exception& ex)
    {
        std::cout << typeid(ex).name() << std::endl;
        std::cout << ex.what() << std::endl;
    }
    // демонстрация #3 работы с exception-ами
    // (свое собственное исключение WrongAddressException)
    // (для примера. Можно было бы спокойно использовать std::invalid_argument)
    try
    {
        booking += "";              // бяка
        std::cout << booking;
    }
    catch (const WrongAddressException& wrong_address)
    {
        std::cout << wrong_address.what() << std::endl;
        std::cout << wrong_address.trace() << std::endl;
    }
    // ПРИМЕЧАНИЯ
    // В конструкторах исключение бросить можно, но объект не создастся до конца!
    // А если не создастся, то какой бы корректный деструктор ни был указан,
    // он просто не вызовется. Поэтому если в конструкторе до генерации исключения
    // будет выделена память, она никогда не освободится
    // (даже если исключение будет перехвачено корректно).
    /*
    Booking::Booking(const std::string addresses[],
                     std::size_t address_count,
                     tm time)
    {
        addresses_ = new std::string [address_count];       // если выделить память здесь, то при
                                                            // выбросе исключения, будет раскручен стек
                                                            // и произойдет мемори лик
        if (address_count < 2)
        {
            throw std::invalid_argument("There should be at least 2 addresses!")
        }
        for (std::size_t i = 0; i < address_count; ++i)
        {
            addresses_[i] = addresses[i];
        }
        address_count_ = address_count;
    }
        // [address_count >= 2]  -  это инвариант (условие в методе, которое должно строго выполняться);
        //                          проверку инварианта можно сделать исключениями или Return Code-ами;
        //                          с помощью assert(address_count >= 2) можно завершить программу при некорректных данных
    */
    // В деструкторах исключения бросать нежелательно!
    // Если исключения все-таки бросаются, то перехватывать нужно внутри же деструктора,
    // иначе оно уже нигде не перехватится, а вызовется завершение программы через std::terminate
    return 0;
}
// функции для демонстрации исключений в начале лекции
int inner_operation(int x, int y)
{
    if (y == 0)
    {
        throw "Zero argument!";     // можно бросать строки
    }
    return x / y;
}
void exception_demo()
{
    int x, y;
    std::cout << "Enter x: ";
```

```
    std::cin >> x;
    if (x == 0)
    {
        throw 0;        // в C++ можно бросать и числа
    }
    std::cout << "Enter y: ";
    std::cin >> y;
    int res = inner_operation(x, y);
    std::cout << "x / y = " << res << std::endl;
}
/*
// Код для демонстрации SEH
// (скомпилируется только майкрософтовским компилятором)
#include <stdexcept>
#include <windows.h>
void SEH_test()
{
    // ааа, это провокация!
    int* p = 0;
    __try
    {
        *p = 42;
    }
    __except(EXCEPTION_EXECUTE_HANDLER)
    {
        std::cout << "AAAAARGH! NULL!!!" << std::endl;
    }
    // обычным траем ничего не поймаем
}
*/
// еще интересный момент:
// catch (Exception& ex) -> все равно копия!
// почитать еще материал, так или иначе касающийся safe code:
// https://cwe.mitre.org/data/index.html  -  Code weaknesses
```

Лекция 9. Шаблонные функции и классы, идиома SFINAE.

- Макросы
- Шаблонные функции
- Шаблонные классы, особенности наследования и static
- Явная специализация шаблона, частичная специализация шаблона
- SFINAE (**S**ubstitution **F**ailure **I**s **N**ot **A**n **E**rror)
- Traits
- Метапрограммирование

Примеры макросов (в том числе забавных) можно найти в файле macros.h.
Примеры шаблонных функций и классов, явной и частичной специализации шаблонов, трейтов и метапрограммирования с помощью шаблонов, находятся в файле templates.h.
В рамках проекта TaxiService написан шаблонный класс-контейнер **DaytimeCollection**. Это такой контейнер, который позволяет хранить любые объекты, но с автоматической их группировкой по времени суток (утренние, дневные, вечерние). Например, класс какого-то события, мероприятия, спортивного матча, заказа и т.д. Объект должен быть типа, предоставляющего метод std::tm getTime().

```
// демонстрация шаблонного контейнера DaytimeCollection

std::tm time;
time.tm_min = 30;
time.tm_sec = 45;

time.tm_hour = 20;
Booking b1("Olimpiyskii", "Planetarium", time);

time.tm_hour = 16;
Booking b2("Cosmos Circus", "Kirshavel", time);

time.tm_hour = 22;
Booking b3("DonNU", "DonNMU", time);

time.tm_hour = 6;
```

```
Booking b4("Donbass Arena", "Railway Station", time);


DaytimeCollection<Booking> bookings;

bookings.add(b1);
bookings.add(b2);
bookings.add(b3);
bookings.add(b4);

std::cout << "=== Morning bookings: ===" << std::endl << std::endl;
bookings.printMorningEntries();

std::cout << "=== Day bookings: ===" << std::endl << std::endl;
bookings.printDayEntries();

std::cout << "=== Night bookings: ===" << std::endl << std::endl;
bookings.printNightEntries();
```

Чтобы этот код скомпилировался, тип T должен иметь метод std::tm T::getTime(). [Есть более серьезные и правильные SFINAE-based способы проверки того, что тип T содержит конкретный метод, но в данной лекции они не рассматриваются. Например, Эрб Саттер предлагает трейты]

```
template <typename T>
bool startsWith(T* arr, T elem)
{
    return arr[0] == elem;
}
// ===================== явная специализация для символов
// ===================== (добавлен функционал регистронезависимости)
template <>
bool startsWith(char* arr, char elem)
{
    return ::toupper(arr[0]) == ::toupper(elem);
}
// ============================ пример шаблонного класса
template <class T, int N>
class Container
{
public:
    void print()
    {
        std::cout << N << " " << typeid(T).name() << std::endl;
    }
protected:
    T a_[N];
};
// ======================== пример №1 частичной специализации
template <int N>
class Container<int, N>
{
public:
    void print()
    {
        std::cout << N << " integers!" << std::endl;
    }
protected:
    int a_[N];
};
//
// Generic programming
// SFINAE       Substitution
//              Failure
//              Is
//              Not
//              An
//              Error
//
// ======================== пример №2 частичной специализации
```

```
template <typename T>
struct Get
{
    const static int Size = sizeof(T);
};
// для всех указателей компилятор будет подставлять именно эту версию структуры
template <typename T>
struct Get<T*>
{
    const static int Size = 0;
    void printYear()
    {
        T obj;
        // и только уже дойдя до этого места, компилятор
        // выдаст ошибку, если у типа T нет age
        std::cout << "Year: " << obj.getYear() << std::endl;
    }
};
// Своя версия трейтов
template <class T>
struct type_trait
{
    const static bool isIntegral = false;
};
template <>
struct type_trait<int>
{
    // typedef нужен для SFINAE (вводится только в int-овой и short-овой специализациях)
    typedef int integral_type;
    // эту константу также можно использовать в каких-либо целях
    const static bool isIntegral = true;
};
template <>
struct type_trait<short>
{
    typedef short integral_type;
    const static int isIntegral = true;
};
// данная функция будет считаться компилятором подходящей только для тех типов,
// в которых объявлен тип integral_type
template <typename T>
void printNumber(T elem, typename type_trait<T>::integral_type flag = 0)
{
    std::cout << elem << std::endl;
}
// Примечание: ноль в суффиксе "= 0" не играет никакой роли (можно хоть 500 написать)
// Главное здесь - в сигнатуру метода добавить нечто сигнальное, чтобы компилятор не пропустил
// в случае нарушения каких-то условий. С другой стороны, чтобы printNumber() можно
// было вызывать без второго чисто вспомогательного параметра, мы ему даем значение по умолчанию.
// ====================== Еще примеры шаблонов и метапрограммирования
// ========================================== Ex.1 (classic example)
/* this is taken when I is even */
template<int I>
void center(char(*)[I % 2 == 0] = 0)
{
    std::cout << "EVEN!" << std::endl;
}
/* this is taken when I is odd */
template<int I>
void center(char(*)[I % 2 == 1] = 0)
{
    std::cout << "ODD!" << std::endl;
}
// ================================== Ex.2 (even more classic example)
template <int N>
struct Factorial
{
    enum { value = N * Factorial<N - 1>::value };
```

```
};
template <>
struct Factorial<0>
{
    enum { value = 1 };
};
class Book
{
public:
    Book() : year_(2000), name_("")
    {
    }
    short getYear() const
    {
        return year_;
    }
protected:
    std::string name_;
    short year_;
};
```

Лекция 10. Шаблонные функции и классы, идиома SFINAE.

- Контейнеры, адаптеры
- Итераторы
- Алгоритмы
- Функторы
- Рецепты STL

STLdemo

Подпроект, в котором демонстрируется применение и особенности последовательностных и ассоциативных контейнеров, алгоритмов и функторов, предоставляемых библиотекой STL. Помимо совсем базовых затронуты такие вопросы, как:

- Адаптеры контейнеров std::stack, std::queue, std::priority_queue
- Подробно об итераторах, трюки с итераторами и std::back_inserter
- Словарь на основе хеш-таблицы std::unordered_map с примером реализации своего типа в качестве ключа
- Особая специализация вектора std::vector<bool> и тип std::bitset
- Идиома erase-remove
- Немножко синтаксиса C++11

В STL есть огромное число готовых (и весьма оптимизированных) функций; целесообразно применять их. Например, числа от 1 до 10 можно вывести с помощью STL, не написав ни строчки цикла или ветвления:

```
std::vector<int> numbazz(10);
std::iota(numbazz.begin(), numbazz.end(), 1);
std::for_each(numbazz.begin(), numbazz.end(), print);
```

TaxiService

Функционал системы не изменялся. Массивы в классах Booking и TaxiService заменены на std::vector. Соответственно, вся работа с коллекциями ведется теперь через методы std::vector и алгоритмы из <algorithm>(остались также и old-school-циклы, но переведены на синтаксис c++11; в методе findDriver закралась лямбда, хотя это тема уже лекции №12). Что ж, код стал короче. Рассмотрен вопрос vec.at(i) vs. vec[i] с точки зрения исключений (см. код перегруженного оператора+= в классе Booking).

```
{
    // 1) КОНТЕЙНЕРЫ
    // последовательные контейнеры STL: vector, deque, list
    std::cout << "vector demo: " << std::endl;
    std::vector<int> v{2, 7, 4, -3, -4, 1};
    /*
     * Как выше, увы, в C++ до стандарта 11 написать нельзя
     *
     * Поэтому в старом C++ пишут просто:
     *
    std::vector<int> v;
    v.push_back(2);
    v.push_back(7);
    v.push_back(4);
    v.push_back(-3);
    v.push_back(-4);
    v.push_back(1);
```

```
*/
std::deque<int> dq;
dq.push_front(7);
std::list<int> il;
il.push_back(12);
// по вектору можно пройтись, как по массиву,
// т.к. у него итератор - произвольный и перегружен оператор []
for (std::size_t i = 0; i < v.size(); ++i)
{
    std::cout << v[i] << " ";
}
std::cout << "Reserve 100 elements!" << std::endl;
v.reserve(100);
std::cout << "Size: " << v.size() << std::endl;
std::cout << "Capacity: " << v.capacity() << std::endl;
std::cout << std::endl;
// адаптеры контейнеров: stack, queue, priority_queue
std::cout << "stack demo: " << std::endl;
std::stack<double, std::vector<double>> s;
s.push(12);
s.push(4);
s.push(5);
std::cout << s.top() << " ";
s.pop();
std::cout << s.top() << " ";
s.pop();
std::cout << std::endl << std::endl;
std::cout << "queue demo: " << std::endl;
std::queue<double> q;
q.push(12);
q.push(4);
q.push(5);
std::cout << q.front() << " " << q.back() << std::endl;
q.pop();
std::cout << q.front() << " " << q.back() << std::endl;
q.pop();
std::cout << std::endl;
std::cout << "priroity queue demo: " << std::endl;
std::priority_queue<double> pq;
pq.push(12);
pq.push(4);
pq.push(5);
std::cout << pq.top() << std::endl;
pq.pop();
std::cout << pq.top() << std::endl;
pq.pop();
std::cout << std::endl << std::endl;
// 2) ИТЕРАТОРЫ
/* Iterators:
 *      Random access iterators:    ++, --, [] + arithmetics, read, write
 *      Bidirectional iterators:    ++, --, read, write
 *      Forward iterators:          ++, read, write
 *      Input iterators:            ++, read
 *      Output iterators            ++, write
 *
 *      Reverse iterators
 *
 *      Advanced:
 *          back_inserter, front_inserter, inserter
 *          istream_iterator, ostream_iterator
 */
std::cout << "iterators demo: " << std::endl;
std::vector<int>::iterator it;
for (it = v.begin(); it != v.end(); ++it)
{
    std::cout << *it << " ";
}
std::cout << std::endl;
```

```
std::cout << "reverse iterator demo: " << std::endl;
typedef std::vector<int>::reverse_iterator moonwalker;
for (moonwalker walker = v.rbegin(); walker != v.rend(); ++walker)
{
    std::cout << *walker << " ";
}
std::cout << std::endl << std::endl;
//std::vector<Person> subscribers;      // композиция
std::vector<Person*> subscribers;       // агрегация
subscribers.push_back(new Person("Ivanov", 20));
subscribers.push_back(new Person("Petrov", 21));
// если делаем агрегацию, то чистим память сами
std::vector<Person*>::iterator pit;
for (pit = subscribers.begin(); pit != subscribers.end(); ++pit)
{
    delete *pit;
}
subscribers.clear();
// впрочем, чтобы это удаление постоянно не писать, можно использовать умные указатели
std::copy(v.begin(), v.end(), std::ostream_iterator<int32_t>(std::cout, " "));
std::vector<double> vcopy;
std::copy(v.begin(), v.end(), std::back_inserter(vcopy));
std::ofstream file("out.txt");
std::copy(vcopy.begin(), vcopy.end(), std::ostream_iterator<double>(file, " "));
file.close();
std::cout << std::endl;
std::ifstream input("out.txt");
std::vector<int> fv((std::istream_iterator<int>(input)),
                    std::istream_iterator<int>());
std::copy(fv.begin(), fv.end(), std::ostream_iterator<double>(std::cout, " "));
std::cout << std::endl;
// 3) АЛГОРИТМЫ И ФУНКТОРЫ
std::cout << "algorithm demo (count_if): " << std::endl;
// алгоритм в функциональном стиле
std::cout << std::count_if(v.begin(), v.end(), isNegative) << " negatives" << std::endl;
// алгоритм в смеси функционального с ООП-стилем (функция завернута в объект-функтор)
NegativeCounter counter;
std::cout << std::count_if(v.begin(), v.end(), counter) << " negatives" << std::endl;
// алгоритмы можно запускать и на обычных массивах
int arr[] = {1, 2, 3, -5, -6, -7, 4, 3, 2, -10};
std::cout << std::count_if(arr, arr + 10, NegativeCounter()) << " negatives" << std::endl;
// Можно свой функтор и не писать, если есть его стандартный аналог:
std::cout << std::count_if(arr, arr + 10, std::bind2nd(std::less<int>(), 0)) << " negatives" << std::endl;
/* std functors:
 *  less, greater, equal_to
 *  minus, plus, divides, modulus, multiplies
 *  logical_not, logical_and, logical_or
 *  bind1st, bind2nd, not1, not2
 *  mem_fun, fun_ptr
 */
std::size_t cnt = std::count_if(
            v.begin(), v.end(), std::bind2nd(std::less<int>(), 3));  // x < 3
std::cout << std::endl << cnt << std::endl;
std::vector<std::string> ws;
std::for_each(ws.begin(), ws.end(), std::mem_fun_ref(&std::string::clear));
std::vector<int> tv(v.size());
std::vector<int> cv(v.size());
std::transform(v.begin(), v.end(), tv.begin(), std::bind2nd(std::multiplies<int>(), 2));
std::copy(tv.begin(), tv.end(), cv.begin());
// Прежде чем писать свои циклы, пройтись по мануалу STL -
// а вдруг там это уже есть.
// Например, там есть даже генерация натуральных чисел от x до y:
std::vector<int> numbazz(10);
std::iota(begin(numbazz), end(numbazz), 1);
std::for_each(numbazz.begin(), numbazz.end(), print);
// iterator.begin()   -    old C++
// begin(iterator)    -    C++11
std::cout << std::endl;
```

```
    // IDIOMS:
    // 1) remove element by value
    tv.erase(std::remove(tv.begin(), tv.end(), 2), tv.end());
    std::copy(tv.begin(), tv.end(), std::ostream_iterator<double>(std::cout, " "));
    // 2) remove unique elements
    std::sort(cv.begin(), cv.end());
    cv.erase(std::unique(cv.begin(), cv.end()), cv.end());
    std::copy(cv.begin(), cv.end(), std::ostream_iterator<double>(std::cout, " "));
    // more:
    // std::next_permutation, std::prev_permutation
    // nth-element, merge, partition
    // ============================================== vector<bool> and bitset demo
    std::vector<bool> hehehe(5, true);
    hehehe[2] = false;
    for (auto b : hehehe)
    {
        std::cout << b << " ";
    }
    std::cout << std::endl;
    std::bitset<7> bits;
    bits.set(1);
    bits.set(5);
    std::cout << bits << std::endl;
    auto xl =  bits.to_ullong();
    std::cout << xl << std::endl;
}
```

Лекция 11. Умные указатели, идиома RAII.

- Resource Acquisition Is Initialization
- Собственный класс "безопасного" файла SafeFile
- Пример собственной реализации умного указателя SafePtr
- std::unique_ptr, std::shared_ptr, std::weak_ptr
- Передача умных указателей в метод/функцию

main.cpp

```
int main()
{
    // продемонстрируем наш безопасный файл
    {
        SafeFile f("a.txt");
        printf("enter string:\n");
        char buf[255];
        std::cin.getline(buf, 255);
        if (strlen(buf) > 7)
        {
            printf("STOP!\n");
            return 1;
        }
        // и здесь должен корректно закрыться
        f.write(buf);
    }
    // и здесь должен корректно закрыться
    // демонстрация нашего умного указателя SafePtr
    int* p = new int(10);
    // ... традиционная работа с указателем
    delete p;   // но всегда надо писать этот дилит ((
    SafePtr<int> iptr(new int(5));
    {
        SafePtr<int> tmp(new int(10));
        std::cout << *tmp << std::endl;
    }
    // утечки памяти не будет, т.к. дилит вызовется в деструкторе
    std::cout << *iptr << std::endl;
    // демонстрация перегрузки оператора->
    SafePtr<Point> pptr(new Point(12, 17));
    std::cout << pptr->x_ << " " << pptr->y_ << std::endl;
    // smart pointers (C++11):
```

```
    // unique_ptr, shared_ptr, make_shared, weak_ptr, lock
    // семантика единоличного владения ресурсом:
    std::unique_ptr<int> uptr1(new int(900));
    std::cout << *uptr1 << std::endl;
    std::unique_ptr<int> uptr2 = std::move(uptr1);
    std::cout << *uptr2 << std::endl;
    // семантика разделяемого владения ресурсом с подсчетом ссылок (use_count)
    std::shared_ptr<int> sp = std::make_shared<int>(1200);
    // use_count = 1
    {
        std::shared_ptr<int> copy = sp; // use_count++  (use_count = 2)
    }
    // use_count--     (use_count = 1)
    std::shared_ptr<int> copy = sp;                     // use_count = 2
    std::cout << *sp << " " << *copy << std::endl;
    std::cout << sp.use_count() << std::endl;
    copy.reset();           // delete int
    // use_count = 1
    std::cout << sp.use_count() << std::endl;
    // не надо так, а то вылетите:
    // std::cout << *copy << std::endl;
    // семантика слабой ссылки
    std::weak_ptr<int> wp;
    {
        std::shared_ptr<int> sptr = std::make_shared<int>(500);
        // use_count = 1
        wp = sptr;
        // use_count = 1
    }
    try
    {
        std::shared_ptr<int> shared_leto = wp.lock();
        // хаха, Shared Leto = 0
        // камеди отдыхаеть
        if (!wp.expired())
        {
            std::cout << *shared_leto << std::endl;
        }
        else
        {
            std::cout << "Pointer expired!" << std::endl;
        }
        // вот тут бросится исключение
        std::shared_ptr<int> s(wp);
    }
    catch (std::bad_weak_ptr& ex)
    {
        std::cout << ex.what() << std::endl;
        return 1;
    }
    // был еще такой auto_ptr, но о нем можно забыть, ибо:
    // std::vector<std::auto_ptr<Point>> vec;
    // vec.push_back(std::auto_ptr<Point>(new Point(1, 2)));
    // std::auto_ptr<Point> p = vec[0];                 // move
    //
    // // work with p
    // std::cout << vec[0]->x_ << std::endl;
    // // delete
    return 0;
}

        template <class T>
        class SafePtr
        {
        public:
            explicit SafePtr(T* p) : ptr_(p)
            {
            }
```

```
    ~SafePtr()
    {
        delete ptr_;
    }
    T* get()
    {
        if (!ptr_) throw 0;             // throw BadPointer
        return ptr_;
    }
    T& operator*()
    {
        return *get();
    }
    T* operator->()
    {
        return get();
    }
private:
    T* ptr_;
private:
    SafePtr(const SafePtr& p);
    SafePtr& operator=(const SafePtr& p);
};
```

Лекция 1. Введение в Python

- Установка Python / anaconda, Python 2.7 и Python 3.5
- IDE: Spyder, PyCharm
- Интерпретатор(ы) Python
- Python Virtual Machine
- Синтаксис Python

```
# работа с консолью
print('Hello')

# в Python 2 можно писать без скобок:
# print 'Hello'

name = input('What is your name?\n')
print('Nice to meet you, ' + name)

# в Python 2 использовать raw_input:
# name = raw_input('What is your name?')
```

```
# Python - язык со строгой динамической типизацией:
x = 42
print(x)
print(type(x))

x = 'Now I am a string!'
print(x)
print(type(x))

# Типы данных (и к ним еще будем возвращаться):
# int, long, str, complex, float, bool, list, tuple, dict, etc.
```

Все дальнейшее изложение относится к реализации CPython.
В Python любая переменная - по сути указатель на блок памяти. Этот блок называется объектом, и в нем хранится (как минимум) счетчик ссылок на объект refcount и тип данных объекта.

```
# ниже выделяется ячейка с содержимым 42, и x ссылается на нее
x = 42
# y указывает на ту же область памяти, что и x
y = x
# после следующей строчки x будет указывать на другой объект:
x = 15
# а y - по-прежнему на 42
print('x=', x, sep='')      # угадайте что такое sep :)
print('id:', id(x))
print('y=', y, sep='')
print('id:', id(y))

# в этой строчке вычислится результат,
# выделится новая память, и x будет ссылаться на нее
x += 10
print('x=', x, sep='')
print('id:', id(x))
print('y=', y, sep='')
print('id:', id(y))

# а объекты с refcount=0 очистятся сборщиком мусора Python
# (впрочем, не всегда!) (см. ниже)
```

Во втором случае выводится True, хотя мы ожидаем False (ведь указатель y не был явно "проброшен" на x).
Что же happened? CPython кэширует в памяти объекты базовых типов (в частности, числа от -5 до 256 (так решили разработчики)). Поэтому при установке значения y=73 интерпретатор установит ссылку на уже имеющуюся ячейку с содержимым 73.
Рассмотрим пример с числами больше 256:

```
x = 273
y = 273
print()
print('x == y :', x == y)
print('x и y ссылаются на одно и то же:', x is y)
```

Таким образом, ячейка с содержимым 73 никогда не очистится сборщиком мусора в CPython, т.к. с точки зрения интерпретатора она должна быть "готова" в любой момент принять на себя ссылку при установке значения другой переменной.
Существует также оператор del:

```
del x
```

В результате данного вызова переменная x будет не определена, а в соответствующем объекте будет уменьшен refcount на 1.
Более подробную и точную информацию по внутренностям CPython можно найти в интернете.

```
# вводим строку, пытаемся преобразовать в int;
# если не получается, ловим исключение, а также
# демонстрируем работу блока finally
# (подробнее об исключениях - в другой лекции)
try:
    age = int(input('How old are you?\n'))
except ValueError as e:
    print('Wrong age!', e)
    # exit()
finally:
    # этот код выполнится, даже если исключение будет перехвачено
```

```
    # (перед exit)
    print("Finally: you'll always see this message")

age += 1

# возможен C-подобный синтаксис форматирования
строки
print("Next year you'll be %d years old" % age)
# далее будем придерживаться, в основном, такого
синтаксиса:
print("Next year you'll be {} years old".format(age))

# пример генерации исключения (raise):
# if age > 90:
#     raise ValueError
```

В примерах выше мы использовали некоторые глобальные функции Python:
id() - получение идентификатора объекта памяти
range() - получение некоторой перечисляемой сущности (об этом позже). Например, range(5) в Python2 вернет список [0,1,2,3,4], в Python3 - т.н. генератор
len() - получение размера коллекции
enumerate() - функция, позволяющая получать на каждой итерации цикла как сам элемент, так и его индекс (а точнее - эта функция возвращает объект, генерирующий пары (индекс, значение))
Еще примеры:

```
# Понятный код
print(max(2, 5))
print(min([5, 3, 2, 7]))
print(sum([10, 20, 30]))

# Финальный case-study:
# вычислим 100!
# 100! = 1 * 2 * 3 * 4 * 5 * ... * 100

factorial = 1
for i in range(1, 101):
    factorial *= i

print('100! =', factorial)

# найдем количество нулей в конце числа 100!
s = str(factorial)
zerocount = 0

# перебрать элементы коллекции задом наперед
# (идиоматичный вариант)
for i in reversed(s):
    if i != '0':
        break
    zerocount += 1

# можно еще так, например (разговор о срезах еще
впереди)
# for i in s[::-1]:
#     ...

print('Количество нулей в конце 100! =
{}'.format(zerocount))
```

```
100! =
933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000
Количество нулей в конце 100! = 24
```

```
# дзен питона
import this
```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those

```
# данные операции работают со списками
# (код станет окончательно понятен в следующей лекции)
print(sorted([10, 5, 3, 7, 1, 8]))
print(list(reversed([1, 2, 3, 4])))
```
[1, 3, 5, 7, 8, 10]
[4, 3, 2, 1]
In [24]:
```
# интересная фича пайтона: цепочки логических
операторов
freq = 11025
if 20 <= freq <= 20000:
    print('Frequency: {} Hz. You can hear it!'.format(freq))
```
Frequency: 11025 Hz. You can hear it!
In [25]:
```
# условное присваивание (тернарный оператор)
mb_size = 690
disc_type = 'CD' if mb_size <= 700 else 'DVD'
print(disc_type)
```
CD
In [26]:
```
# можно писать цепочки таких операторов
mb_size = 4500
disc_type = 'CD' if mb_size <= 700 else 'DVD' if mb_size <=
4700 else '?'
print(disc_type)
```

Лекция 2. Типы данных в Python

В лекции будут рассмотрены самые распространенные и полезные встроенные типы-коллекции:

- изменяемые (mutable)
  - списки (list)
  - словари (dict)
  - множества (set)
- неизменяемые (immutable):
  - строки (str)
  - кортежи (tuple)
  - фиксированные множества (frozenset)

Списки

```
In [1]:
# примеры списков:
letters = ['H', 'e', 'l', 'l', 'o']
stats = [2, 36, 154, 18, 24, 56]
params = [1, 'Normal', [30, 2, 6], 9.82]

print(type(params))
print(params)
print('Третий элемент списка:', params[2])
<class 'list'>
[1, 'Normal', [30, 2, 6], 9.82]
Третий элемент списка: [30, 2, 6]
In [2]:
# списки являются изменяемыеми,
# т.е. можно написать так:
params[1] = 'High'
print(params)
[1, 'High', [30, 2, 6], 9.82]
In [3]:
# список чисел от 1 до 10
l = list(range(1, 11))
print(l)

# список чисел от 10 до 100 с шагом 10
t = list(range(10, 101, 10))
print(t)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

NB. В Python 3 нужно результат функции range() явно преобразовать к списку (к этому важному моменту еще вернемся). В Python 2 функция range() сразу возвращает готовый список, а эквивалент третьепитоновской range() в Python 2 - функция xrange()

```
In [4]:
# узнать размер списка:
print(len(l))

# эквивалентно magic-методу __len__
print(l.__len__())
10
10
In [5]:
# проверить, входит ли элемент в список:
x = 9
if x in l:
    print('Список содержит число x=9')

# эквивалентно magic-методу __contains__
print(l.__contains__(x))
Список содержит число x=9
True
In [6]:
# примеры слайсов (срезов)
print(l[2:])
print(l[:-2])
print(l[1:6:2])
print(l[::-1])
[3, 4, 5, 6, 7, 8, 9, 10]
[1, 2, 3, 4, 5, 6, 7, 8]
[2, 4, 6]
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
In [7]:
# копирование списка
lcopy = l[:]
lcopy[1] = 'Something new'

print('New list:', lcopy)
print('Old list:', l)
New list: [1, 'Something new', 3, 4, 5, 6, 7, 8, 9, 10]
Old list: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
In [8]:
# слайсы + изменяемость списков = ♥
l[3:7] = ['wow', 'what', 'a', 'feature']
print(l)

l[7:7] = ['values', 'were', 'inserted!']
print(l)
[1, 2, 3, 'wow', 'what', 'a', 'feature', 8, 9, 10]
[1, 2, 3, 'wow', 'what', 'a', 'feature', 'values', 'were', 'inserted!', 8,
9, 10]
In [9]:
# пример удаления
del l[3]
print(l)

del l[3:5]
print(l)
[1, 2, 3, 'what', 'a', 'feature', 'values', 'were', 'inserted!', 8, 9, 10]
[1, 2, 3, 'feature', 'values', 'were', 'inserted!', 8, 9, 10]
In [10]:
# склеивание списков
spring = ['march', 'april', 'may']
summer = ['june', 'july', 'august']

# (при этом вызывается magic-метод: __add__)
seasons = spring + summer
print(seasons)

# эквивалентно:
seasons = spring.__add__(summer)
print(seasons)
['march', 'april', 'may', 'june', 'july', 'august']
['march', 'april', 'may', 'june', 'july', 'august']
In [11]:
# операция * (размножение списка)
months = spring * 3
print(months)

years = [2014] * 3 + [2015] * 3 + [2016] * 3
print(years)
['march', 'april', 'may', 'march', 'april', 'may', 'march', 'april',
'may']
[2014, 2014, 2014, 2015, 2015, 2015, 2016, 2016, 2016]
In [12]:
# Получим список пар вида "месяц_год" за период 2014-
2016 гг.

# Данный код приводит к нужному результату, но
питонисты так не пишут:
```

```
spring_2014_16 = []
for i in range(len(years)):
    spring_2014_16.append('{}_{}'.format(months[i], years[i]))
print(spring_2014_16)
['march_2014', 'april_2014', 'may_2014', 'march_2015',
'april_2015', 'may_2015', 'march_2016', 'april_2016',
'may_2016']
In [13]:
# А вот это по-питоньячему:
# применяем list comprehension (генератор списка)
# о функции zip позже - в лекции №4
spring_2014_16 = ['{}_{}'.format(month, year) for month, year
in zip(months, years)]
print(spring_2014_16)
['march_2014', 'april_2014', 'may_2014', 'march_2015',
'april_2015', 'may_2015', 'march_2016', 'april_2016',
'may_2016']
In [14]:
# еще примеры лист компрехеншнов:
cubes = [x**3 for x in range(1, 6)]
print(cubes)

first_ones = [x*10 if x % 2 == 1 else x for x in range(1, 11)]
print(first_ones)

ascii_codes = [(code, chr(code)) for code in range(58, 68)]
print(ascii_codes)
[1, 8, 27, 64, 125]
[10, 2, 30, 4, 50, 6, 70, 8, 90, 10]
[(58, ':'), (59, ';'), (60, '<'), (61, '='), (62, '>'), (63, '?'), (64, '@'),
(65, 'A'), (66, 'B'), (67, 'C')]
In [15]:
row = [[1, 2, 3]]
matrix = row * 3
print(matrix)
[[1, 2, 3], [1, 2, 3], [1, 2, 3]]
In [16]:
# но есть нюанс:
matrix[0][1] = 100
print(matrix)

# как правило, матрицы нужны в математике,
# а там стандартно используется numPy
[[1, 100, 3], [1, 100, 3], [1, 100, 3]]
In [17]:
# можно и стандартными средствами достичь ожидаемого
результата
# (опять с помощью генератора списка)
matrix = [[1,2,3] for _ in range(3)]
print(matrix)
print()

matrix[0][1] = 100
print(matrix)
[[1, 2, 3], [1, 2, 3], [1, 2, 3]]

[[1, 100, 3], [1, 2, 3], [1, 2, 3]]
In [18]:
# демонстрация некоторых функций списков:

l = list(range(10, 16))
print(l)

l.append(7)
print('Добавили 7: ', l)

l.extend([8, 9, 10])
print('Расширили на 8,9,10: ', l)

l.sort()
print('Отсортировали: ', l)
# функция sorted предпочтительнее
print('Отсортировали: ', sorted(l))

tens = l.count(10)
print('Количество десяток: ', tens)

l.clear()
print('Очистили: ', l)
[10, 11, 12, 13, 14, 15]
Добавили 7:  [10, 11, 12, 13, 14, 15, 7]
Расширили на 8,9,10:  [10, 11, 12, 13, 14, 15, 7, 8, 9, 10]
Отсортировали:  [7, 8, 9, 10, 10, 11, 12, 13, 14, 15]
Отсортировали:  [7, 8, 9, 10, 10, 11, 12, 13, 14, 15]
Количество десяток:  2
Очистили:  []
In [19]:
# сравнение списков через == является глубоким,
поэлементным
l1 = [1, 3, [5, 7]]
l2 = [1, 3, [5, 7]]

if l1 == l2:
    print('Списки эквивалентны')
Списки эквивалентны
```

Кортежи

```
In [21]:
# пример кортежа (tuple)
player1 = (1, "Ivanov", 1986)
player2 = 2, "Petrov", 1987     # можно без скобок

print(type(player1))

team = [player1, player2]     # список из кортежей
for player in team:
    print(player)

# Индексация и срезы - как и в списках
print(player1[1])
print(player1[2])
<class 'tuple'>
(1, 'Ivanov', 1986)
(2, 'Petrov', 1987)
Ivanov
1986
Технически отличие кортежей от списков заключается в их
неизменяемости, т.е. так написать нельзя:
player1[2] = 1990
Слайсы есть, но по вышеприведенной причине только на
чтение
In [22]:
# Благодаря кортежам, можно компактно записывать:

# множественное присваивание
x, y = (12, 35)
x, y = 12, 35

# обмен значениями переменных (python-стайл)
x, y = y, x
In [23]:
# В Python 3 есть удобный синтаксис распаковки:

x, *y = range(5)
print(x)
print(y)
0
[1, 2, 3, 4]
In [24]:
first, *middle, last = ['A', 'O', 'E', 'I', 'U']
```

```
print(first)
print(middle)
print(last)
A
['O', 'E', 'I']
U
In [25]:
# кейс-стади со списком и кортежами:
# отсортировать список игроков по году рождения
# в порядке убывания
team.sort(key=lambda x: x[2], reverse=True)
print(team)

# второй вариант (более предпочтительный)
from operator import itemgetter
print(sorted(team, key=itemgetter(2), reverse=True))

# в обоих случаях использованы пока незнакомые lambda и
import
# в следующих лекциях они станут знакомы ))
[(2, 'Petrov', 1987), (1, 'Ivanov', 1986)]
[(2, 'Petrov', 1987), (1, 'Ivanov', 1986)]
In [26]:
# выше мы рассматривали пример со списком ascii-кодов.
```

Строки

По поводу строк поговорим еще в лекции №8, а пока рассмотрим базовый функционал. Строки, как и кортежи, являются immutable.

```
In [28]:
# однострочные и многострочные литералы
message = 'Hello world'
print(type(message))
print(message)

text = '''He deals the cards as a meditation
And those he plays never suspect
He doesn't play for the money he wins
He don't play for respect'''
print(text)
<class 'str'>
Hello world
He deals the cards as a meditation
And those he plays never suspect
He doesn't play for the money he wins
He don't play for respect
In [29]:
# еще такой вариант разбивки строк
text = ('He deals the cards as a meditation\n'  # каждую строку
можно комментить
        'And those he plays never suspect\n'    # автор -
М.Г.Саммнер
        'He doesn\'t play for the money he wins\n'
        'He don\'t play for respect\n')
print(type(text))
print(text)
<class 'str'>
He deals the cards as a meditation
And those he plays never suspect
He doesn't play for the money he wins
He don't play for respect

In [30]:
# следует аккуратно экранировать символы
path = 'D:\Folder\file.txt'
print(path)
path = 'D:\\Folder\\file.txt'
print(path)
D:\Folderile.txt
D:\Folder\file.txt
In [31]:
# или использовать raw-строки (r)

path = r'D:\Folder\file.txt'
print(path)
D:\Folder\file.txt
```

Как уже отмечалось в лекции №1, есть два способа форматирования строк. В наших лекциях постоянно присутствует то или иное форматирование.
Ну а подробно - см. здесь:
https://docs.python.org/3.5/library/stdtypes.html#printf-style-string-formatting
https://docs.python.org/3.5/library/string.html#formatstrings

```
In [32]:
# как и со списками - размножение и конкатенация
hidden = '?' * 16
print(hidden)

hidden += ' (unknown)'
print(hidden)
????????????????
???????????????? (unknown)
In [33]:
# привычный сплит
s = 'one two three four five'
numbers = s.split()
print(numbers)
['one', 'two', 'three', 'four', 'five']
In [34]:
# удобный джойн
sentence = ' and '.join(numbers)
print(sentence)
one and two and three and four and five
In [35]:
# пример - заменить все цифры на символ "*" (без
использования регулярок)
info = 'my phone no.: 050-888-77-65'

# 1 вариант
secret = ''.join(['*' if c.isdigit() else c for c in info])
print(secret)

# 2 вариант
secret = info.translate(str.maketrans('0123456789',
'**********'))
print(secret)
my phone no.: ***-***-**-**
my phone no.: ***-***-**-**
In [36]:

# Напишем то же самое, но в круглых скобках:
ascii_codes = ((code, chr(code)) for code in range(58, 68))

# при выводе не отобразится содержимое,
# т.к. в даном случае создается не список и даже не кортеж,
# а "генератор", о котором речь пойдет в лекции №4
print(ascii_codes)

# генератор "поставляет" данные по мере итерации по нему
# (что весьма эффективно)
for code in ascii_codes:
    print(code)
<generator object <genexpr> at 0x02D64330>
(58, ':')
(59, ';')
(60, '<')
(61, '=')
(62, '>')
(63, '?')
(64, '@')
(65, 'A')
(66, 'B')
(67, 'C')
```

```
# пример: в строках C++ кода вычленить только списки параметров функций
# (опять без регулярок)
cpp_code = ('void foo(int* array, size_t N);'
            'int bar(std::string s);'
            'bool compare(double x, double y);')

code_lines = cpp_code.rstrip(';').split(';')

parameter_lists = [line[line.index('(')+1:line.index(')')]
                   for line in code_lines]

for parameter_list in parameter_lists:
    print('Function parameters:', parameter_list)
```

```
Function parameters: int* array, size_t N
Function parameters: std::string s
Function parameters: double x, double y
```

Словари

In [38]:

```
# пример словаря
cities = {'Donetsk': 1869, 'Moscow': 1147}

print(type(cities))
print(cities)
print(cities['Donetsk'])
```

```
<class 'dict'>
{'Moscow': 1147, 'Donetsk': 1869}
1869
```

In [39]:

```
# Добавим элемент в список
cities['Lugansk'] = 1795

print(cities)
```

```
{'Moscow': 1147, 'Lugansk': 1795, 'Donetsk': 1869}
```

In [40]:

```
# Отдельно отобразим ключи и значения словаря
for city in cities.keys():
    print(city)

for fdate in cities.values():
    print(fdate)
```

```
Moscow
Lugansk
Donetsk
1147
1795
1869
```

In [41]:

```
# пример работы со строкой и словарем:
# разобьем строку на слова
# и посчитаем число вхождений каждого в строку
text = "skating away on the thin ice of a new day away"

words = text.split(' ')
print(words)

# следующий код работает, но он не идиоматичен:
hist = {}
for word in words:
    if word in hist:     # из-за проверки на вхождение в словарь
        hist[word] += 1
    else:
        hist[word] = 1

print(hist)
```

```
['skating', 'away', 'on', 'the', 'thin', 'ice', 'of', 'a', 'new', 'day', 'away']
{'new': 1, 'skating': 1, 'a': 1, 'thin': 1, 'on': 1, 'day': 1, 'away': 2, 'the': 1, 'of': 1, 'ice': 1}
```

In [42]:

```
# ниже то же самое, но по-питоновски
# проверки нет, есть метод get(key, default)
hist = {}
for word in words:
    hist[word] = hist.get(word, 0) + 1

print(hist)

# также можно использовать Counter из модуля collections
```

```
{'new': 1, 'skating': 1, 'a': 1, 'thin': 1, 'on': 1, 'day': 1, 'away': 2, 'the': 1, 'of': 1, 'ice': 1}
```

In [43]:

```
# и опять же есть аналог лист компрехеншна:
ascii_codes = { code: chr(code) for code in range(58, 68) }
print(ascii_codes)
```

```
{64: '@', 65: 'A', 66: 'B', 67: 'C', 58: ':', 59: ';', 60: '<', 61: '=', 62: '>', 63: '?'}
```

Лекция 3. Модули и пакеты в Python

- подключение модулей
- интроспекция модулей
- собственные модули и пакеты
- примеры стандартных модулей:
    - math
    - random
    - datetime, time, calendar
    - re
    - itertools
    - collections

```
# посмотрим значения первых "особенных" членов модуля math:
# строка документации модуля (docstring)
print('__doc__:', math.__doc__)

# квалифицированное имя модуля
print('__name__:', math.__name__)

# пакет, которому принадлежит модуль
print('__package__:', math.__package__)

# служебный объект-загрузчик модуля (начиная с Python 3.3)
print('__loader__:', math.__loader__)
```

```
# служебная информация загрузчика модуля (начиная с Python 3.4)
# https://docs.python.org/3/reference/import.html#import-related-module-attributes
print('__spec__:', math.__spec__)
__doc__: This module is always available.  It provides access to the
mathematical functions defined by the C standard.
__name__: math
__package__:
__loader__: <class '_frozen_importlib.BuiltinImporter'>
__spec__: ModuleSpec(name='math', loader=<class '_frozen_importlib.BuiltinImporter'>, origin='built-in')
```

Любой *.py скрипт является модулем, равно как и текущая сессия интерпретатора. Т.е. мы, как и в предыдущем примере, можем посмотреть значения служебных переменных в текущем модуле (или выполняемой сессии интерпретатора):

In [8]:

```
print('__doc__:', __doc__)
print('__name__:', __name__)
print('__package__:', __package__)
print('__loader__:', __loader__)
print('__spec__:', __spec__)
__doc__: Automatically created module for IPython interactive environment
__name__: __main__
__package__: None
__loader__: None
__spec__: None
```

Обратите внимание на значение __name__. Если модуль запущен интерпретатором, то значение __name__ = '__main__'. Если модуль импортируется, то __name__ будет совпадать с именем модуля (например, 'mymodule.py').

Важно: при импорте модуля Python выполнит все инструкции, которые определены в этом модуле (при множественном импорте одного и того же модуля, тем не менее, инструкции будут выполнены только один раз).

Иногда модуль проектируется как для использования его функций и констант, так и для автономного запуска (например, для какой-либо демонстрации его возможностей или специфических тестов). Тогда в коде модуля пишется стандартная проверка:

In [9]:

```
# здесь описываем функции и константы модуля
# ...

if __name__ == '__main__':
    # здесь прописать те инструкции, которые
    # должны выполняться при автономном запуске скрипта
    print('This script is executing right now')
This script is executing right now
```

Модули можно структурировать по пакетам. Подробно здесь:
https://docs.python.org/3/tutorial/modules.html#packages

Если кратко, то нужно соблюсти определенную структуру директорий со скриптами:

```
sound/                          пакет верхнего уровня
      __init__.py               инициализация пакета sound
      formats/                  суб-пакет для преобразований форматов файлов
            __init__.py
            wavread.py
            wavwrite.py
            aiffread.py
            aiffwrite.py
            ...
      effects/                  суб-пакет для звуковых эффектов
            __init__.py
            echo.py
            surround.py
            reverse.py
            ...
```

Файлы __init__.py нужны для того, чтобы Python воспринимал директорию как контейнер пакета.

Сами эти файлы могут быть пустыми, но также можно определить там некоторые константы и функции, общие для всего пакета.

Кроме того, можно задать значение переменной __all__ (о ней - ниже).

Пример импорта суб-пакета:

```
import sound.effects.echo
```

Внутрипакетные ссылки

Если пакеты содержат суб-пакеты (как в случае с пакетом-примером sound), то для обращения к суб-пакетам используется абсолютный импорт. Например, если модулю sound.filters.vocoder нужен доступ к функциям модуля echo из пакета sound.effects, надо прописать

```
from sound.effects import echo
```

Можно также использовать относительный импорт. При этом для описания текущего и родительского пакетов используется символ точки. Например, для модуля surround можно написать:

```
from . import echo
```

from .. import formats
from ..filters import equalizer
Относительное импортирование опирается на имя текущего модуля. Т.к. имя главного модуля всегда __main__, то модули, используемые в качестве главных модулей приложения, должны импортироваться "абсолютным" способом.

Некоторые замечания по локации модулей. При импорте модуля интерпретатор Python ищет данный модуль в следующей последовательности:
Текущая директория
Если модуль не найден, Python ищет его в каждой директории из списка переменной среды PYTHONPATH.
Если модуль по-прежнему не найден, Python проверяет путь по умолчанию. Например, в UNIX-системах, это обычно /usr/local/lib/python/.
Переменная PYTHONPATH содержит список директорий. В Windows прописать какой-либо путь в PYTHONPATH можно так:
set PYTHONPATH=c:\python\lib;
в UNIX-системах так:
set PYTHONPATH=/usr/local/lib/python
Строка пути поиска модуля хранится в системном модуле sys в переменной sys.path. Эта переменная содержит полный путь текущей директории, значение переменной PYTHONPATH и путь по умолчанию, выставленный при установке интерпретатора Python.

Лекция 4. Элементы функционального программирования в Python
- синтаксис функций
- LEGB
- итераторы и генераторы
- лямбда-функции
- filter, map, reduce, zip
- модуль itertools (is back)
- декораторы
- модуль functools

LEGB (области видимости)
L, Local — все локальные имена: имена, любым способом присвоенные внутри функции (объявленных как def или lambda), и не помеченные ключевым словом global в этой функции.
E, Enclosing function locals — все имена в локальной области видимости всех замыкающих функций (объявленных как def или lambda), от самой вложенной ко внешней.
G, Global (на уровне модуля) — имена, глобальные для модуля или помеченные ключевым словом global внутри конструкции def где-либо в файле.
B, Built-in (Python) — Предопределенные глобальные имена Python, которые списком содержатся в переменной __builtins__ (которые, в свою очередь, берутся из модуля builtins): print, open, reversed, ValueError и т.д.

```
x = 73

def wrong_scope():
    print(x)    # это сработает
    x += 1      # здесь бросит исключение

wrong_scope()
---------------------------------------------------------------------------
UnboundLocalError
in wrong_scope()
      2
      3 def wrong_scope():
----> 4     print(x)    # это сработает
      5     x += 1      # здесь бросит исключение
      6

UnboundLocalError: local variable 'x' referenced before
assignment
In [22]:
x = 73

def scope_ok():
    global x
    print(x)
    x += 1

scope_ok()

print(x)
73
74
In [23]:
# более полная демонстрация LEGB, функции locals(),
# а также замыканий

x = 42
y = 73


# замыкающая функция
def enclosing_func():
    x = 500
    y = 1000
    print('Enclosing function locals:', locals())

    # внутренняя (вложенная) функция
    def inner_func():
        global x
        print('Inner func x, y:', x, y)
        print('Inner func locals:', locals())
        x += 1

    return inner_func

func = enclosing_func()
func()

print('Global x, y:', x, y)
Enclosing function locals: {'y': 1000, 'x': 500}
Inner func x, y: 42 1000
Inner func locals: {'y': 1000}
Global x, y: 43 73
In [24]:
# демонстрация использования ключевого слова nonlocal:

def enclosing_func():
```

```
    value = 0
    # вложенная функция наращивает value замыкающей
функции
    def inner_func():
        nonlocal value
        value += 1
        return value
    return inner_func

id_generator = enclosing_func()

print(id_generator())
print(id_generator())
```
1
2
In [25]:
```
# Версия Python 2 (где нет nonlocal)

# для эмуляции nonlocal можно завести переменную
mutable типа,
# например, список из одного элемента:

def enclosing_func():
    value = [0]
    def inner_func():
        value[0] += 1
        return value[0]
    return inner_func

id_generator = enclosing_func()

print(id_generator())
print(id_generator())
```
1
2
In [26]:
```
n = 1

# переменные for-циклов "протекают" в глобальное
пространство имен
for n in range(5):
    print(n, '(в цикле)')

print(n, '(в глобальной области видимости)')
```
0 (в цикле)
1 (в цикле)
2 (в цикле)
3 (в цикле)
4 (в цикле)
4 (в глобальной области видимости)
In [27]:
```
# В данном случае "протекания" не будет:

i = 1
print([i for i in range(5)])
print(i, '(в глобальной области видимости)')
```
[0, 1, 2, 3, 4]
1 (в глобальной области видимости)
Итераторы и генераторы:
итератор - это концепция (любой объект, имеющий методы
next() и __iter__())
генератор - языковое средство (объект вокруг функции с
yield)
Любой генератор является итератором, но не наоборот.
In [28]:
```
def odd_iterator(x):
    for i in range(1, x, 2):
        yield i

for x in odd_iterator(10):
    print(x)
```
1
3
5
7
9
In [29]:
```
it = odd_iterator(10)
it
```
Out[29]:
<generator object odd_iterator at 0x02D7A450>
In [30]:
```
print(next(it))
print(next(it))
```
1
3
In [31]:
```
try:
    it.throw(ValueError('Iterator problems!'))
except ValueError as err:
    print('Caught:', err)

it.close()
```
Caught: Iterator problems!

```
# Синтаксис *args, **kwargs:
def pretty_print(text, *args, **kwargs):
    # отобразить "шапку"
    for cnt in args:
        print(kwargs['fill'] * cnt)

    # вывести собственно текст
    print(text)

    # отобразить футер, если параметр footer=True
    if kwargs['footer']:
        for cnt in reversed(args):
            print(kwargs['fill'] * cnt)

    # args kwargs
    print()
    print('args:')
    print(args)
    print('kwargs:')
    print(kwargs)


pretty_print('Hello', 20, 10, 5, fill='~', footer=True)
```

```
# таким образом, 1) функции можно присваивать
переменным:
func = foo
func(100)
```
Out[15]:
110
In [16]:
```
# 2) можно делать массив из функций
#    и последовательно выполнять их:
import math

funcs = [foo, math.factorial]

for func in funcs:
    print(func.__qualname__, func(5))
```
foo 15
factorial 120
In [17]:
```
# 3) можно передавать функцию в функцию:
def process_elementwise(seq, func):
    return [func(s) for s in seq]
```

```
words = ['Fundamentals', 'of', 'brainwashing...']
process_elementwise(words, str.__len__)
Out[17]:
[12, 2, 15]
In [18]:
process_elementwise(words, str.upper)
Out[18]:
['FUNDAMENTALS', 'OF', 'BRAINWASHING...']
In [19]:
nums = [1, -3, 5, -7, -9]
process_elementwise(nums, abs)
Out[19]:
[1, 3, 5, 7, 9]
In [20]:
process_elementwise(nums, foo)
Out[20]:
[11, 7, 15, 3, 1]

# Но это еще не все:
# есть также метод send, который позволяет делать т.н.
корутины
def coroutine_example():
    while True:
        x = yield
        print('Got new number:', x)

cor = coroutine_example()
next(cor)

cor.send(2)
cor.send(5)
cor.send(10)

cor.close()
Got new number: 2
Got new number: 5
Got new number: 10
In [35]:
# предыдущий пример был игрушечный, но в асинхронных
фреймворках
# корутины являются важнейшим элементом, и в них
можно писать
# концептуально что-то вроде этого (чтобы это работало,
нужен
# постоянно крутящийся цикл, в котором данные
асинхронно
# получаются и передаются, подробнее - в лекции №11):

# здесь логика генерации данных для обработки
def number_provider():
    for i in range(1, 11):
        yield i

# здесь логика получения данных и их обработки
def number_processor():
    while True:
        x = yield from number_provider
        print('Got new number:', x)
In [36]:
# в Python 3.3 появилась конструкция yield from:

def odd_number_provider(n):
    for i in range(1, n, 2):
        yield i

def even_number_provider(n):
    for i in range(2, n, 2):
        yield i

def number_generator(n):
    yield from odd_number_provider(n)
    yield from even_number_provider(n)

for number in number_generator(10):
    print(number)
1 3 5 7 9 2 4 6 8

In [37]:
# еще пример генератора (генерирующего цифры числа)
def generate_digits(x):
    while x > 0:
        yield x % 10
        x = x // 10

for digit in generate_digits(5716):
    print(digit)
6
1
7
5
In [38]:
l = [x for x in generate_digits(54321)]
l
Out[38]:
[1, 2, 3, 4, 5]
In [39]:
i = generate_digits(4523)
print([next(i), next(i), next(i)])
[3, 2, 5]
In [40]:
# еще пример генератора (генерирующего перестановки n
чисел)
import itertools

def permute(n):
    for perm in itertools.permutations(range(1, n+1)):
        yield perm

for p in permute(3):
    print(p)
(1, 2, 3)
(1, 3, 2)
(2, 1, 3)
(2, 3, 1)
(3, 1, 2)
(3, 2, 1)
In [41]:
# еще раз о разнице между range в python2 и python3
nums = range(1, 5)
nums

# в Python2 будет выведено [1, 2, 3, 4]
# аналог: nums = xrange(1, 5)

Лямбда-функции
In [43]:
# рассмотрим пример функции, которая преобразует
# количество секунд в формат времени (например,
звучания трека)

def duration_string(seconds):
    return '{}:{:02d}'.format(seconds // 60, seconds % 60)

print(duration_string(411))
print(duration_string(120))
6:51
2:00
In [44]:
# небольшие функции можно объявлять "на ходу"
# это анонимные функции (лямбда-функции)

dur = lambda seconds: '{}:{:02d}'.format(seconds // 60,
seconds % 60)
```

```
print(dur(411))
print(dur(120))
6:51
2:00
In [45]:
type(dur)
Out[45]:
function
In [46]:
print(dur.__name__)
print(dur.__code__.co_varnames)
<lambda>
('seconds',)
```
Часто лямбда-функции используются в качестве параметров функций map, filter, reduce
```
In [47]:
# демонстрация filter()

files = ['1.wav', '2.mp3', '3.jpg', '4.png', '5.wav']

sound_files = filter(
   lambda f: f.endswith('wav') or f.endswith('mp3'), files)

print(sound_files)
print(list(sound_files))
<filter object at 0x02D8AA10>
['1.wav', '2.mp3', '5.wav']
In [48]:
# первым параметром функции filter() является любая функция;
# если функция используется одноразово и/или короткая телом,
# то лямбда хорошо подходит, иначе - подавать готовую функцию

def is_sound_file(filename):
   return filename.endswith('wav') or filename.endswith('mp3')

list(filter(is_sound_file, files))
Out[48]:
['1.wav', '2.mp3', '5.wav']
In [49]:
# эквивалент filter() в виде генератора списка:
sound_files = [f for f in files
            if f.endswith('wav') or f.endswith('mp3')]

sound_files
Out[49]:
['1.wav', '2.mp3', '5.wav']
In [50]:
# конечно, можно написать и так:
sound_files = []
for f in files:
   if f.endswith('wav') or f.endswith('mp3'):
      sound_files.append(f)

sound_files

# НО НЕ НАДО, это ж не по-питоновски! ))
Out[50]:
['1.wav', '2.mp3', '5.wav']
In [51]:
# демонстрация функции map()

# 1)
extensions = map(
   lambda f: f[f.index('.')+1:], files)

extensions = list(extensions)
print(extensions)

# 2)
print(list(map(str.upper, extensions)))
['wav', 'mp3', 'jpg', 'png', 'wav']
['WAV', 'MP3', 'JPG', 'PNG', 'WAV']
In [52]:
# эквивалент map() в виде генератора списка

# 1)
extensions = [f[f.index('.')+1:] for f in files]
print(extensions)

# 2)
extensions = [ext.upper() for ext in extensions]
print(extensions)
['wav', 'mp3', 'jpg', 'png', 'wav']
['WAV', 'MP3', 'JPG', 'PNG', 'WAV']
In [53]:
# пример: отобразить числа в интервалы соответствующей длины

lengths = [3, 5, 1, 2]

intervals = map(lambda n: tuple(range(1, n+1)), lengths)
intervals = list(intervals)

print(intervals)
print()
for interval in intervals:
   print(interval)
[(1, 2, 3), (1, 2, 3, 4, 5), (1,), (1, 2)]

(1, 2, 3)
(1, 2, 3, 4, 5)
(1,)
(1, 2)
In [54]:
# Еще пример: посчитаем сумму цифр числа в декларативном стиле:
n = 123
sum(map(int, str(n)))
Out[54]:
6
In [55]:
# NB. Функция reduce() в Python 2 является встроенной,
#     в Python 3 - кроется в модуле functools

from functools import reduce

s = range(1, 5)
reduce(lambda x, y: x + y, s)
Out[55]:
10
In [56]:
# что здесь происходит?
words = ['He', 'evolves', 'fast', 'slowly']

sentence = reduce(
   lambda x, y: x + ' ' + y if x[-1] == y[0] else x, words)
sentence
Out[56]:
'He evolves slowly'
```
Еще элементы декларативного программирования в Python
```
In [57]:
# демонстрация функции zip()

name = ['John', 'Pete', 'Sue', 'Bob', 'Alice']
sex = ['m', 'm', 'f', 'm', 'f']
age = [27, 21, 25, 24, 23]

students = zip(name, sex, age)
print(students)
students = list(students)
print(students)
```

```
<zip object at 0x02D9DE40>
[('John', 'm', 27), ('Pete', 'm', 21), ('Sue', 'f', 25), ('Bob', 'm', 24),
('Alice', 'f', 23)]
In [58]:
# удобные и полезные функции any() и all():
help(any)
help(all)
Help on built-in function any in module builtins:

any(iterable, /)
    Return True if bool(x) is True for any x in the iterable.

    If the iterable is empty, return False.

Help on built-in function all in module builtins:

all(iterable, /)
    Return True if bool(x) is True for all values x in the iterable.

    If the iterable is empty, return True.

In [59]:
any(s[2] > 25 for s in students)
Out[59]:
True
In [60]:
all(s[2] < 25 for s in students)
Out[60]:
False
In [61]:
all(s[2] < 30 for s in students)
Out[61]:
True
In [62]:
print(max(s[2] for s in students))
print(min(s[0] for s in students))
27
Alice
```

Продолжим рассмотрение возможностей модуля itertools, начатое в предыдущей лекции

```
In [63]:
from itertools import islice

list(islice(students, 1, 3))
Out[63]:
[('Pete', 'm', 21), ('Sue', 'f', 25)]
In [64]:
from itertools import takewhile

some = takewhile(lambda s: len(s[0]) > 3, students)
list(some)
Out[64]:
[('John', 'm', 27), ('Pete', 'm', 21)]
In [65]:
from itertools import dropwhile

some = dropwhile(lambda s: len(s[0]) > 3, students)
list(some)
Out[65]:
[('Sue', 'f', 25), ('Bob', 'm', 24), ('Alice', 'f', 23)]
In [66]:
from itertools import groupby

students = sorted(students, key=lambda s: s[1])

for sex, info in groupby(students, lambda s: s[1]):
    print(list(info))
[('Sue', 'f', 25), ('Alice', 'f', 23)]
[('John', 'm', 27), ('Pete', 'm', 21), ('Bob', 'm', 24)]
```

В данном файле приводится сравнение кодов решения задач с помощью Python, C# LINQ и Java 8 Streams:

Декларативная парадигма в Python, C# и Java (слайды)
Содержимое слайдов подробно разбирается на лекции.
Декораторы

```
In [67]:
def copyright(func):

    def _wrapper():
        func()
        print('(c) Wise man')

    # декоратор возвращает функцию (callable)
    return _wrapper


# декорируем функцию
@copyright
def print_slogan():
    print('The truth will set you free')


print_slogan()

# происходит такой вызов:
# copyright(print_slogan())()
The truth will set you free
(c) Wise man
In [68]:
# пример декоратора функции, имеющей параметры:

def stringify(func):

    def _wrapper(*args, **kwargs):
        # преобразуем каждый аргумент функции к строке
        str_args = [str(arg) for arg in args]
        # вызовем исходную функцию со строковыми
аргументами
        return func(*str_args, **kwargs)

    return _wrapper


# декорируем функцию
@stringify
def concat(x, y):
    return x + y


concat('Result: ', 10)

# происходит такой вызов:
# stringify(concat('OK', 10))()

# если убрать декоратор @stringify, при вызове будет
ошибка
Out[68]:
'Result: 10'
In [69]:
# Однакож:
print(concat.__name__)
print(concat.__qualname__)

# хотелось бы увидеть тут 'concat'...
_wrapper
stringify.<locals>._wrapper
In [70]:
# можно сделать что-то вроде этого:

def decorate(func):
    def _wrapped(*args, **kwargs):
        return func(args, kwargs)

    # явно скопировать необходимые переменные
```

```
    _wrapped.__name__ = func.__name__

    return _wrapped
```

```
@decorate
def baz():
    return 'baz'
print(baz.__name__)
```

Подытожим:
База функционального программирования:
чистые функции (pure functions) - детерменированные функции без побочных эффектов
lambda-функции
функции - это first-class объекты
неизменяемость объектов и потоков данных
рекурсии
ленивые вычисления (в т.ч. генераторы)
map, filter, reduce

Лекция 5. Работа с файлами и файловой системой
- работа с файлами
- модули os, shutil
- модули glob, pathlib
- сериализация: pickle, JSON
- ZIP-архивация, модуль zipfile

```
s = ['Hello ', 'World\n', '(c) Admin\n']

# запись списка строк в текстовый файл demo.txt:
try:
    f = open(r'data\demo.txt', 'wt')
    f.writelines(s)
except IOError as err:
    print(err)
except:
    print('Something\'s gone wrong...')
else:
    print('File\'s been updated succesfully')
finally:
    if f:
        f.close()
File's been updated succesfully
In [8]:
# смысл конструкции with - более короткая запись кода:
#
# -- setup --
# try:
#     -- do_smth --
# finally:
#     -- tear down --
#

# чтение строк из файла (с конструкцией with)
try:
    with open(r'data\demo.txt', 'rt') as f:
        lines = f.readlines()
except IOError as err:
    print(err)
    lines = []

# вывести строки из файла в "сыром" виде
print('Read from file:', lines)


# 1 - убрать \n на конце каждой строки с помощью rstrip и
map
newlines = list(map(lambda x: x.rstrip(), lines))
# 2 - убрать \n на конце каждой строки с помощью rstrip и
list comprehension
newlines = [x.rstrip() for x in lines]

print('After post-processing:', newlines)
Read from file: ['Hello World\n', '(c) Admin\n']
After post-processing: ['Hello World', '(c) Admin']
In [9]:
file = open(r'data\demo.txt', 'r+')
file.seek(6)
word = file.read(9);
print(word)

pos = file.tell();
print('Current file position:', pos)

file.close()
World
(c)
Current file position: 16
Работа с файловой системой и ОС
In [10]:
import os

os.getcwd()
Out[10]:
'C:\\Users\\Tim\\Documents\\Python Scripts'
In [11]:
# забавно создаем пустой файл:
open(r'data\ghost.txt', 'wt').close()

# переименуем
os.rename(r'data\ghost.txt', r'data\to_delete.txt')

# да и удалим
os.remove(r'data\to_delete.txt')
Модуль send2trash отвечает за безопасное удаление
файлов:
import send2trash

send2trash.send2trash('data\demo.txt')
In [12]:
os.path.exists('What.txt')
Out[12]:
False
In [13]:
os.path.isdir('data')
Out[13]:
True
In [14]:
os.path.isfile('Useful Resources.ipynb')
Out[14]:
True
In [15]:
os.path.join('D:\\', 'Program Files', 'Projects')
Out[15]:
'D:\\Program Files\\Projects'
In [16]:
```

```
os.path.basename(os.getcwd())
Out[16]:
'Python Scripts'
In [17]:
os.path.dirname(os.getcwd())
Out[17]:
'C:\\Users\\Tim\\Documents'
In [18]:
'{} bytes'.format(os.path.getsize('Useful Resources.ipynb'))
Out[18]:
'28801 bytes'
```

Еще часть функций из модуля os:

os.listdir
os.mkdir
os.makedirs
os.system - устарело, используется subprocess (см. следующую лекцию)

```
In [19]:
os.listdir(os.getcwd())
Out[19]:
['.git',
 '.gitignore',
 '.ipynb_checkpoints',
 'algo',
 'data',
 'lec01 - Intro.ipynb',
 'lec02 - Data types.ipynb',
 'lec03 - Modules and packages.ipynb',
 'lec04 - Declarative Python, CSharp, Java.pdf',
 'lec04 - Elements of functional programming.ipynb',
 'lec05 - Working with Files.ipynb',
 'lec06 - Processes and IPC.ipynb',
 'README.md',
 'Useful Resources.ipynb']
In [20]:
# получить список только папок:
folders = [entry for entry in os.listdir(os.getcwd())
           if os.path.isdir(entry)]
folders
Out[20]:
['.git', '.ipynb_checkpoints', 'algo', 'data']
In [21]:
COPY_DIR = r'D:\PythonCourse'

files = os.listdir(os.getcwd())
file_copies = [os.path.join(COPY_DIR, f) for f in files]
file_copies
Out[21]:
['D:\\PythonCourse\\.git',
 'D:\\PythonCourse\\.gitignore',
 'D:\\PythonCourse\\.ipynb_checkpoints',
 'D:\\PythonCourse\\algo',
 'D:\\PythonCourse\\data',
 'D:\\PythonCourse\\lec01 - Intro.ipynb',
 'D:\\PythonCourse\\lec02 - Data types.ipynb',
 'D:\\PythonCourse\\lec03 - Modules and packages.ipynb',
 'D:\\PythonCourse\\lec04 - Declarative Python, CSharp,
Java.pdf',
 'D:\\PythonCourse\\lec04 - Elements of functional
programming.ipynb',
 'D:\\PythonCourse\\lec05 - Working with Files.ipynb',
 'D:\\PythonCourse\\lec06 - Processes and IPC.ipynb',
 'D:\\PythonCourse\\README.md',
 'D:\\PythonCourse\\Useful Resources.ipynb']
In [22]:
try:
    os.mkdir('test1')
    os.makedirs(r'test2\we_need\to_go_deeper')
except Exception as err:
    print(err)
In [23]:
try:
    os.rmdir('test1')
    os.rmdir(r'test2\we_need\to_go_deeper')
    # shutil.rmtree('test2')
except Exception as err:
    print(err)
In [24]:
folder = os.getcwd()

# итератор, рекурсивно проходящий по директории
dir_iterator = os.walk(folder)

for i in range(4):
    print('Iteration {}:\n'.format(i+1))

    root, dirs, files = next(dir_iterator)

    # исключим служебные папки, начинающиеся с точки
    dirs[:] = [d for d in dirs if not d[0] == '.']

    print('Root:', root)
    print('Subfolders:', dirs)
    print('Files:', files)
    print()

Iteration 1:

Root: C:\Users\Tim\Documents\Python Scripts
Subfolders: ['algo', 'data', 'test2']
Files: ['.gitignore', 'lec01 - Intro.ipynb', 'lec02 - Data
types.ipynb', 'lec03 - Modules and packages.ipynb', 'lec04 -
Declarative Python, CSharp, Java.pdf', 'lec04 - Elements of
functional programming.ipynb', 'lec05 - Working with
Files.ipynb', 'lec06 - Processes and IPC.ipynb', 'README.md',
'Useful Resources.ipynb']

Iteration 2:

Root: C:\Users\Tim\Documents\Python Scripts\algo
Subfolders: ['__pycache__']
Files: ['search.py', 'sort.py', '__init__.py']

Iteration 3:

Root: C:\Users\Tim\Documents\Python
Scripts\algo\__pycache__
Subfolders: []
Files: ['search.cpython-35.pyc', '__init__.cpython-35.pyc']

Iteration 4:

Root: C:\Users\Tim\Documents\Python Scripts\data
Subfolders: []
Files: ['data.json', 'data.pkl', 'demo.txt', 'demo.xml',
'democopy.txt', 'temp.txt']
```

Сериализация / десериализация

```
In [34]:
# модуль pickle (сериализация / десериализация любых
объектов)
import pickle

data = (1, 3.15, 'Hello', [1, 4, 5])

# сериализация
pickle.dump(data, open(r'data\data.pkl', 'wb'))

# десериализация
obj = pickle.load(open(r'data\data.pkl', 'rb'))
print(obj)
(1, 3.15, 'Hello', [1, 4, 5])
In [35]:
import simplejson
```

```
# json-сериализация
with open(r'data\data.json', 'w') as f:
    simplejson.dump(data, f, indent=4)

# json-десериализация
json = simplejson.load(open(r'data\data.json', 'r'))
print(json)
[1, 3.15, 'Hello', [1, 4, 5]]
In [36]:
with open(r'data\data.json', 'r') as f:
    for line in f:
        print(line.rstrip())
[
    1,
    3.15,
    "Hello",
    [
        1,
        4,
        5
    ]
]
In [37]:
# XML
# Пример взят из документации:
# https://docs.python.org/3.5/library/xml.etree.elementtree.html

xml = '''<?xml version="1.0"?>
<data>
    <country name="Liechtenstein">
        <rank>1</rank>
        <year>2008</year>
        <gdppc>141100</gdppc>
        <neighbor name="Austria" direction="E"/>
        <neighbor name="Switzerland" direction="W"/>
    </country>
    <country name="Singapore">
        <rank>4</rank>
        <year>2011</year>
        <gdppc>59900</gdppc>
        <neighbor name="Malaysia" direction="N"/>
    </country>
    <country name="Panama">
        <rank>68</rank>
        <year>2011</year>
        <gdppc>13600</gdppc>
        <neighbor name="Costa Rica" direction="W"/>
        <neighbor name="Colombia" direction="E"/>
    </country>
</data>
'''

xmlfile = open(r'data\demo.xml', 'w')
xmlfile.write(xml)
xmlfile.close()
In [38]:
import xml.etree.ElementTree as ET

tree = ET.parse(r'data\demo.xml')
root = tree.getroot()
root.tag
Out[38]:
'data'
In [39]:
for child in root:
    print(child.tag, child.attrib)
country {'name': 'Liechtenstein'}
country {'name': 'Singapore'}
country {'name': 'Panama'}
In [40]:
# демонстрация адресации XPath
root.findall(".")
Out[40]:
[<Element 'data' at 0x02CBB5A0>]
In [41]:
neighbors = root.findall('./country/neighbor')
for neighbor in neighbors:
    print(neighbor.get('name'))
Austria
Switzerland
Malaysia
Costa Rica
Colombia
In [42]:
root.findall(".//year/..[@name='Singapore']")
Out[42]:
[<Element 'country' at 0x02D77D80>]
In [43]:
print(root.findall(".//*[@name='Singapore']/year")[0].text)
2011
In [44]:
second_neighbors = root.findall(".//neighbor[2]")
for neighbor in second_neighbors:
    print(neighbor.get('name'))
Switzerland
Colombia
In [45]:
# Еще модуль с DOM-парсером:
import xml.dom.minidom


DOMTree = xml.dom.minidom.parse(r'data\demo.xml')
collection = DOMTree.documentElement
print('Root element: {}'.format(collection.tagName))
Root element: data
In [46]:
countries = collection.getElementsByTagName("country")

for country in countries:
    print("Country: {}".format(country.getAttribute('name')))

    print('Neighbors:')
    neighbors = country.getElementsByTagName('neighbor')
    for neighbor in neighbors:
        print('\t', neighbor.getAttribute('name'))
Country: Liechtenstein
Neighbors:
        Austria
        Switzerland
Country: Singapore
Neighbors:
        Malaysia
Country: Panama
Neighbors:
        Costa Rica
        Colombia
In [47]:
# ZIP-архивация
# Подробнее здесь: https://pymotw.com/3/zipfile/
import zipfile
import datetime

files = [r'data\data.json', r'data\data.pkl', r'data\demo.txt']

print('crating arhive...')

archive = zipfile.ZipFile(r'data\archive.zip', mode='w')
try:
    for file in files:
        print('adding {}'.format(file))
        archive.write(file)
finally:
```

```
    print('closing...\n')
    archive.close()


for info in archive.infolist():
    print(info.filename)
    print('\tComment:\t', info.comment)
    print('\tModified:\t', datetime.datetime(*info.date_time))
    print('\tSystem:\t\t', info.create_system, '(0 = Windows, 3 =
Unix)')
    print('\tZIP version:\t', info.create_version)
    print('\tCompressed:\t', info.compress_size, 'bytes')
    print('\tUncompressed:\t\n', info.file_size, 'bytes')
```

Лекция 6. Работа с процессами и операционной системой

- аргументы командной строки, модуль sys
- модуль argparse
- модуль ConfigParser
- модуль subprocess
- перенаправление потоков ввода-вывода

```
# Далее вызовем сторонний скрипт с аргументами ком.
строки.
# Вот его содержимое:

script = open(r'script\argparse_demo.py', 'rt')

for codeline in script:
    print(codeline.rstrip())
import argparse
import sys


def check_arg(args=None):
    ''' parsing demo function '''

    parser = argparse.ArgumentParser(
        description='Script to learn basic argparse')

    parser.add_argument('-H', '--host',
                        help='host ip',
                        required='True',
                        default='localhost')
    parser.add_argument('-p', '--port',
                        help='port of the web server',
                        default='8080')
    parser.add_argument('-u', '--user',
                        help='user name',
                        default='root')
    parser.add_argument('-m', '--misc',
                        help='miscellaneous',
                        action='store_true')

    parsed = parser.parse_args(args)
    return parsed.host, parsed.port, parsed.user, parsed.misc


if __name__ == '__main__':
    h, p, u, m = check_arg(sys.argv[1:])
    print('host = {}'.format(h))
    print('port = {}'.format(p))
    print('user = {}'.format(u))
    print('misc = {}'.format(m))
In [6]:
%run script\argparse_demo.py --host 127.0.0.1 -u admin -m
host = 127.0.0.1
port = 8080
user = admin
misc = True
In [7]:
# модуль для работы с INI-файлами
import configparser

config = configparser.ConfigParser()

config['NETWORK'] = {'Port': '12087',
                     'MaxConnections': '100',
                     'Redirect': 'No'}

config['LOGGING'] = {}
config['LOGGING']['LogFile'] = 'LOG/history.log'
config['LOGGING']['AutoClean'] = 'Yes'

config['EMAIL'] = {}
email = config['EMAIL']
email['Address'] = 'foo@bar.baz'
email['PeriodDays'] = '7'

with open(r'data\example.ini', 'w') as configfile:
    config.write(configfile)
In [8]:
for codeline in open(r'data\example.ini', 'rt'):
    print(codeline.rstrip())
[NETWORK]
maxconnections = 100
redirect = No
port = 12087

[LOGGING]
logfile = LOG/history.log
autoclean = Yes

[EMAIL]
address = foo@bar.baz
perioddays = 7

In [9]:
config = configparser.ConfigParser()
config.read(r'data\example.ini')
config.sections()
Out[9]:
['NETWORK', 'LOGGING', 'EMAIL']
In [10]:
for key in config['NETWORK']:
    print(key)
maxconnections
redirect
port
In [11]:
config['LOGGING']['logfile']
Out[11]:
'LOG/history.log'
Модуль subprocess
In [12]:
import subprocess

# вызов процесса / консольной команды
# (выводящей на экран переменную PATH)
# т.к. сам echo является не процессом,
# а встроенной командой в shell, то указываем shell=True

subprocess.call(r'echo %PATH%', shell=True)
```

```
# при запуске интерпретатора Python результат отобразится в консоли,
# но в ячейки jupyter-тетрадей автоматически не перенаправится

# вызов внешнего процесса (ipconfig.exe)
# с получением выходного результата (синхронно)

output = subprocess.check_output(['ipconfig', '/all'])

# циферки вам знать не нужно ))
print(output.decode('cp866').translate(
      str.maketrans('01235789', '????????')))
```

```
Настройка протокола IP для Windows

   Имя компьютера  . . . . . . . . . : ARISTO
   Основной DNS-суффикс  . . . . . . :
   Тип узла. . . . . . . . . . . . . : Гибридный
   IP-маршрутизация включена . . . . : Нет
   WINS-прокси включен . . . . . . . : Нет
```

```
%run script\ask.py
What is your name?Joe
Hello, Joe
In [16]:
p = subprocess.Popen([r'python', r'script\ask.py'],
                     stdin=subprocess.PIPE)
p.stdin.write(b'Joe')

# в консоли интерпретатора Python будет нормальный ввод-вывод
Out[16]:
3
In [17]:
# вызов внешнего процесса (ping.exe) и работа с потоком вывода
process = subprocess.Popen(['ping', '127.0.0.1'],
stdout=subprocess.PIPE)

# синхронный busy spin, работа с потоком вывода, пока процесс не отработает
while True:
    output = process.stdout.readline()

    # когда процесс завершится, poll() вернет код завершения, а не None
    rc = process.poll()

    if rc is not None:
        break
    if output:
        print(output.strip().decode('cp866'))

print('The process has finished with return code {}\n'.format(rc))
```

```
Обмен пакетами с 127.0.0.1 по с 32 байтами данных:
Ответ от 127.0.0.1: число байт=32 время<1мс TTL=128
Ответ от 127.0.0.1: число байт=32 время<1мс TTL=128
Ответ от 127.0.0.1: число байт=32 время<1мс TTL=128
Ответ от 127.0.0.1: число байт=32 время<1мс TTL=128

Статистика Ping для 127.0.0.1:
Пакетов: отправлено = 4, получено = 4, потеряно = 0
(0% потерь)
Приблизительное время приема-передачи в мс:
Минимальное = 0мсек, Максимальное = 0 мсек, Среднее = 0 мсек
The process has finished with return code 0

In [18]:
# пример программной реализации вызова цепочки:
# ipconfig /all | findstr IPv4

p1 = subprocess.Popen(['ipconfig', '/all'],
                      stdout=subprocess.PIPE)
p2 = subprocess.Popen(['findstr', 'IPv4'],
                      stdin=p1.stdout,
                      stdout=subprocess.PIPE)
p1.stdout.close()
output = p2.communicate()[0]
print(output.decode('cp866'))
```

```
   IPv4-адрес. . . . . . . . . . . . : 192.168.0.100(Основной)
```

Лекция 7. Объектно-ориентировавнное программирование в языке Python

- особенности инкапсуляции, наследования и полиморфизма в Python
- динамическое создание классов
- метаклассы
- __slots__
- пользовательские исключения
- менеджеры контекста
- глубокое копирование
- декораторы через классы
- класс enum
- модуль attrs

```
from classes.fractions import Fraction

f1 = Fraction(1, 5)
f2 = Fraction()
f2.num = 2
f2.den = 5
print(f2.den)
print(f2['den'])
```

```
5
5
In [8]:
f = f1 + f2  # перегрузка '+'     __add__
print(f)     # перегрузка 'str'   __str__

# т.к. f присваивается новое значение, старый удалится
deleted
3/5
```

```
In [9]:
# перегрузка 'len' __len__
len(f)
Out[9]:
8
In [10]:
# перегрузка 'in' __contains__
if f1 in f2:
    print('sub fraction')
sub fraction
In [11]:
f1 < f2
Out[11]:
True
In [12]:
print('The number of instances is equal to {}'.format(Fraction.count))
```

```
The number of instances is equal to 3
new-style classes
в Python2 наследоваться от object
в Python3-классы - сразу new style (от object наследоваться
все равно желательно)
Новоявленные свойства new-style классов:
низкоуровневые конструкторы __new__()
дескрипторы, обощенный способ настройки доступа к
атрибуту
статические методы и class методы
свойства (вычисляемые атрибуты)
декораторы
слоты
новый порядок Method Resolution Order (MRO)

Наследование и утиная типизация
In [13]:
from classes.labels import Label, Barcode, QRcode

qr = QRcode()
qr.filename = 'aaa.jpg'     # setting property

code = qr.recognize()
print(code)

images = [qr, Barcode(), Label(300, 300)]
for im in images:
    im.scale(100, 100)      # duck typing
The QRcode was scaled at 100 x 100
lalalala
The QRcode was scaled at 100 x 100
The Barcode was scaled at 100 x 100
The Label was scaled at 100 x 100
Что хотим сделать (аналог на C#)
interface IParser
{
    void Load();
    string Parse();
}

class XMLParser : IParser
{
    void Load() { ... }
    string Parse() { ... }
}

class JSONParser : IParser
{
    void Load() { ... }
    string Parse() { ... }
}
In [14]:
from classes.parsers import XMLParser, JSONParser,
StringParser
from classes.parsers import ParserFactory

p1 = JSONParser()
p1.load('a.json')

p2 = XMLParser()
p2.load('a.xml')

parsers = [p1, p2, StringParser('hahaha')]

for parser in parsers:
    parser.parse()        # duck typing
Parsing JSON: a.json
Parsing XML: a.xml
foo... hahaha
In [15]:
# демонстрация ParserFactory
filename = "111.json"
# а потом попробуем
# filename = "111.xml"

factory = ParserFactory()
parser = factory.create_parser(filename)

parser.parse()
Parsing JSON: 111.json
Динамическое создание классов и метаклассы
In [16]:
Person = type('Person', (object,),
              {'name': 'vasya',
               'age': 21,
               'calc': lambda self: self.age + 1})
p = Person()
print(p.name)
print(p.calc())

# class Person(object):
#     def __init__(self):
#         self.name = 'vasya'
#         self.age = 21
#
#     def calc(self):
#         self.age += 1
#

In [17]:
# Пример метакласса
# http://eli.thegreenplace.net/2011/08/14/python-metaclasses-
by-example

class PointMeta(type):
    def __new__(meta, name, bases, dct):
        print('-----------------------------------')
        print('Allocating memory for class', name)
        print(meta)
        print(bases)
        print(dct)
        return super(PointMeta, meta).__new__(meta, name,
bases, dct)

    def __init__(cls, name, bases, dct):
        print('\n-----------------------------------')
        print('Initializing class', name)
        print(cls)
        print(bases)
        print(dct)
        super(PointMeta, cls).__init__(name, bases, dct)

    def __call__(cls, *args, **kwds):
        print('__call__ of ', str(cls))
        print('__call__ *args=', str(args))
        print()
        return type.__call__(cls, *args, **kwds)
In [18]:
class Point(metaclass=PointMeta):

    # в Python 2 пишется здесь:
    #__metaclass__ = PointMeta

    def __init__(self, x, y):
        print('Point p({}, {})'.format(x, y))


pt = Point(42, 73)
-----------------------------------
Allocating memory for class Point
<class '__main__.PointMeta'>
()
```

```
{'__module__': '__main__', '__qualname__': 'Point', '__init__':
<function Point.__init__ at 0x7f70d4535620>}

-----------------------------------
Initializing class Point
<class '__main__.Point'>
()
{'__module__': '__main__', '__qualname__': 'Point', '__init__':
<function Point.__init__ at 0x7f70d4535620>}
__call__ of  <class '__main__.Point'>
__call__ *args= (42, 73)

Point p(42, 73)
__slots__
```

Специальный атрибут __slots__ позволяет явно указать в коде, какие атрибуты экземпляра ожидаются быть в наличии у объекта, со всеми вытекающими результатами:
более быстрый доступ к атрибуту.
потенциальная экономия памяти.
http://stackoverflow.com/questions/472000/usage-of-slots
По умолчанию, экземпляры и старых, и new-style классов хранят атрибуты в словарях. Память расходуется неэффективно, если нужно хранить всего несколько атрибутов. Особенно это становится критичным в случае хранения большого числа таких "скромных" объектов.

Объявление в классе __slots__ резервирует ровно столько памяти под атрибуты, сколько нужно для их хранения.
К примеру, атрибуты ОРМ-ки SQLAlchemy эффективно хранятся в памяти благодаря тому, что реализованы через __slots__.
In [19]:
```
class SmallObject(object):

    __slots__ = ['_width', '_height', '_path']

    def __init__(self):
        self._width = 10
        self._height = 10
        self._path = 'default'

    @property
    def size(self):
        return self._width * self._height

    @property
    def path(self):
        return self._path


obj = SmallObject()
print(obj.path)
print(obj.size)
```
```
default
100
```
Пример пользовательского исключения
In [20]:
```
class FileTooBigError(Exception):

    def __init__(self, filesize):
        super(FileTooBigError, self).__init__(
            'Filesize is too big: {} bytes'.format(filesize))
        self._filesize = filesize

    # если __str__() должен возвращать просто сообщение
message,
    # то метод можно не определять
    # (по умолчанию, строковое представление Exception - и
так message)
    # def __str__(self):
    #     return self.message


import os

filesize = os.path.getsize(r'lec01 - Intro.ipynb')

try:
    if filesize > 4096:
        raise FileTooBigError(filesize)
except FileTooBigError as err:
    print(err)
```
```
Filesize is too big: 34224 bytes
```
Менеджеры контекста
In [21]:
```
# ПЕРВЫЙ СПОСОБ:

class PingContext(object):

    def __enter__(self):
        self._log = open(r'data/log.txt', 'wt')
        print('Log file is ready')
        return self

    def write(self, text):
        self._log.write(text)

    def __exit__(self, exc_type, exc_value, exc_traceback):
        if exc_value != None:
            print('Ping failed! Wrong IP format: %s' % exc_value)
            # подчистить ресурсы, которыми владеет контекст
        self._log.close()
        print('Log file is closed')
        return True
        #return False          # если False, то заново бросается
Exception
                              # (и надо будет перехватывать во
внешнем коде)


IPs = ['127.0.0.1', '0.0.0.0', '1.1.1.1', '10.0.0.1', 'f.2.3.4',
'192.168.0.102']

# регулярка для простейшей проверки валидности IP-
адреса
import re
regex = re.compile(r'\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}')

with PingContext() as log:
    # проходим по всем записям из строк IP-адресов
    for IP in IPs:
        if not re.match(regex, IP):
            raise ValueError(IP)
        # пинг IP-адреса ...
        # и запись в лог
        log.write(IP + '\n')
```
```
Log file is ready
Ping failed! Wrong IP format: f.2.3.4
Log file is closed
```
In [22]:
```
# ВТОРОЙ СПОСОБ:

from contextlib import contextmanager

@contextmanager
def ping_log():
    # __enter__
    print('Log file is ready')
    log = open(r'data/log.txt', 'wt')

    # actions
    try:
        yield log
```

```
    # __exit__
    except ValueError as err:
        print('Ping failed! Wrong IP format: %s' % err)
    finally:
        log.close()
        print('Log file is closed')


regex = re.compile(r'\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}')

IPs = ['127.0.0.1', '0.0.0.0', '1.1.1.1', '10.0.0.1', 'f.2.3.4',
'192.168.0.102']

with ping_log() as log:
    for IP in IPs:
        if not re.match(regex, IP):
            raise ValueError(IP)
        log.write(IP + '\n')
Log file is ready
Ping failed! Wrong IP format: f.2.3.4
Log file is closed
Глубокое копирование
In [23]:
# демо глубокого копирования стандартных типов
import copy

l1 = [1, "Hello", [3,4,5], (2, [7,8,9], 'x'), 12]

l2 = copy.deepcopy(l1)
l2[1] = "World"

print(l1)
print(l2)
[1, 'Hello', [3, 4, 5], (2, [7, 8, 9], 'x'), 12]
[1, 'World', [3, 4, 5], (2, [7, 8, 9], 'x'), 12]
In [24]:
# демо глубокого копирования объектов классов
from fractions import Fraction

f1 = Fraction(3, 5)
f2 = copy.deepcopy(f1)  # дефолтная реализация;
                        # ее можно переопределить в классе Fraction:
                        # метод __deepcopy__(self, f)
print(f1)
print(f2)
Fraction was deleted
Fraction was deleted
3/5
3/5

Декоратор через класс
In [26]:
import functools

class Prettifier(object):
    def __init__(self, char='*'):
        self._char = char

    def __call__(self, func):
        def decorated(*args, **kwargs):
            print(self._char * len(args[0]))
            func(*args, **kwargs)
            print(self._char * len(args[0]))
        return decorated

@Prettifier('=')
def function(text):
    print(text)


function('decoratin\' da text!')
====================
decoratin' da text!
====================
модули enum и attrs
In [27]:
from enum import Enum, unique

@unique
class OperatingSystem(Enum):
    WINDOWS = 0
    UNIX = 1
    MACOS = 2

OperatingSystem.UNIX
Out[27]:
<OperatingSystem.UNIX: 1>
In [28]:
target_system = OperatingSystem.WINDOWS

if target_system == OperatingSystem.WINDOWS:
    print('Windows')
Windows
In [29]:
import attr

@attr.s
class TCPConnection(object):
    ip = attr.ib(default='127.0.0.1')
    port = attr.ib(default=8080)
    proxies = attr.ib(default=attr.Factory(list))


conn = TCPConnection('https://www.python.org',
proxies=['1.1.1.1', '2.2.2.2'])
print(conn.ip)
print(conn.port)
print(conn.proxies)
https://www.python.org
8080
['1.1.1.1', '2.2.2.2']
```

Лекция 8. Работа с бинарными данными и кодировками
- модуль struct
- модуль io и класс BytesIO
- строки и байты

```
In [1]:
import struct

In [2]:
# F.  C Type          Python type         Standard size
# x   pad byte        no value
# c   char            string of length    1
# b   signed char     integer             1
# B   unsigned char   integer             1
# ?   _Bool           bool                1
# h   short           integer             2
# H   unsigned short  integer             2
# i   int             integer             4
# I   unsigned int    integer             4
# l   long            integer             4
# L   unsigned long   integer             4
# q   long long       integer             8
# Q   unsigned long   long  integer       8
# f   float           float               4
# d   double          float               8
```

```
# s   char[]         string
# p   char[]         string
# P   void *         integer


# <     Little endian
# >     Big endian
In [3]:
struct.pack('hhl', 65, 66, 67)
Out[3]:
b'A\x00B\x00C\x00\x00\x00'
In [4]:
struct.pack('BBB', 65, 66, 67)
Out[4]:
b'ABC'
In [5]:
struct.unpack('>hhl', b'\x00\x01\x00\x02\x00\x00\x00\x03')
Out[5]:
(1, 2, 3)
In [6]:
struct.calcsize('hhl')
Out[6]:
8
In [7]:
record = b'Messi     \x0A\x29\xC3\x07'

lastname, no, goals, born = struct.unpack('<10sbbH', record)

print(lastname)
print(no)
print(goals)
print(born)
b'Messi     '
10
41
1987
In [8]:
# Формат BMP
#                               Заголовок файла
#
# 0   2   Символы 'BM' (код 4D42h)
# 2   4   Размер файла в байтах
# 6   2   0 (Резервное поле)
# 8   2   0 (Резервное поле)
# 10  4   Смещение, с которого начинается само
изображение (растр).
#
#                               Заголовок BITMAP (Информация об
изображении)
#
# 14  4   Размер заголовка BITMAP (в байтах) равно 40
# 18  4   Ширина изображения в пикселях
# 22  4   Высота изображения в пикселях
# 26  2   Число плоскостей, должно быть 1
# 28  2   Бит/пиксел

# 1 = monochrome palette.       Кол-во цветов = 2
# 4 = 4bit palletized.          Кол-во цветов = 16
# 8 = 8bit palletized.          Кол-во цветов = 256
# 16 = 16bit RGB.                 Кол-во ветов = 65536
# 24 = 24bit RGB.                 Кол-во ветов = 16M

# 30  4   Тип сжатия

# 0 = BI_RGB                (без сжатия)
# 1 = BI_RLE8               (8 bit RLE сжатие)
# 2 = BI_RLE4               (4 bit RLE сжатие)

# 34  4   0 или размер сжатого изображения в байтах.
# 38  4   Горизонтальное разрешение, пиксел/м
# 42  4   Вертикальное разрешение, пиксел/м
# 46  4   Количество используемых цветов
# 50  4   Количество "важных" цветов.
# Палитра (Карта цветов для N цветов), если используется
# 54  4*N Палитра


with open(r'data\trapeze.bmp', 'rb') as f:
    header = f.read(14)
    bm, size, reserve1, reserve2, relpos = \
        struct.unpack('<2sLHHL', header)
    print('{} bytes'.format(size))

    bitmapheader = f.read(16)
    headersize, width, height, planes, bitsperpixel = \
        struct.unpack('<LLLHH', bitmapheader)
    print('Size: {}x{} pixels'.format(width, height))
    print('{} bits per pixel'.format(bitsperpixel))
750054 bytes
Size: 500x500 pixels
24 bits per pixel
In [9]:
import io

b = io.BytesIO(b"abcdef")

view = b.getbuffer()
view[2:4] = b'56'
b.getvalue()
Out[11]:
b'ab56ef'
In [12]:
b = io.BytesIO(b'\x2F\x87\x5AOK\x9f')
b.seek(3)
b.getvalue()
Out[12]:
b'/\x87ZOK\x9f'
In [13]:
f = io.BytesIO(bytes.fromhex('30'))
f.seek(1)
f.write(bytes.fromhex('ca fe ba be'))
f.write(bytes.fromhex('57 41 56 45'))
f.getvalue()
Out[13]:
b'0\xca\xfe\xba\xbeWAVE'
Системы счисления, дампы
In [14]:
x = 1243
hex(x)
Out[14]:
'0x4db'
In [15]:
oct(x)
Out[15]:
'0o2333'
In [16]:
bin(x)
Out[16]:
'0b10011011011'
In [17]:
hexdump = ' '.join([hex(byte) for byte in b'Hello'])
hexdump
Out[17]:
'0x48 0x65 0x6c 0x6c 0x6f'
In [18]:
hexdump = ' '.join([hex(byte)[2:] for byte in b'Hello'])
hexdump
Out[18]:
'48 65 6c 6c 6f'
In [19]:
a = int('00100001', 2)
a
Out[19]:
33
```

```
In [20]:
a = int('0xff', 16)
a
Out[20]:
255
Строки, байты, юникод
In [21]:
s = 'Строка ёжика'
print(type(s))
print(s)
print(s.encode('utf8').decode('cp1251'))
<class 'str'>
Строка ёжика
РЎС‚СЂРѕРєР° С‘Р¶РёРєР° !!!
In [22]:
b = s.encode('utf8')
print(type(b))
print('UTF8:', b)
print('CP1251:', s.encode('cp1251'))
<class 'bytes'>
UTF8: b'\xd0\xa1\xd1\x82\xd1\x80\xd0\xbe\xd0\xba\xd0\xb0
\xd1\x91\xd0\xb6\xd0\xb8\xd0\xba\xd0\xb0'
CP1251: b'\xd1\xf2\xf0\xee\xea\xe0 \xb8\xe6\xe8\xea\xe0'
In [23]:
# В python 2 ситуация кардинально другая:
# есть отдельные типы str и bytes.

# s = u'Строка ёжика'
# print(type(s))
# print(s)
# print(s.encode('utf8').decode('cp1251'))

# b = s.encode('utf8')
# print(type(b))
# print(b)
```

Лекция 9. Приложения с графическим интерфейсом

- tkInter
- wxPython
- Qt

tkInter:

```
# python 3:
from tkinter import *
from tkinter.messagebox import *

# python 2:
# from Tkinter import *
# from tkMessageBox import *
# from ttk import *


class DiscFrame(Frame):
    ''' The class of the frame that shows user's media collection '''
    def __init__(self, parent):
        Frame.__init__(self, parent)
        self.parent = parent
        self.init_ui()

    def add_disc(self, event):
        ''' Handler of add_button events '''
        if self.check_cd.get() == 1:
            self.cd_listbox.insert(self.cd_listbox.size(),
                                   self.disc_entry.get())
        if self.check_dvd.get() == 1:
            self.dvd_listbox.insert(self.dvd_listbox.size(),
                                    self.disc_entry.get())

    def turn_red(self, event):
        ''' Handler of help_button events '''
        event.widget["foreground"] = "red"
        showinfo(self.disc_entry.get(),
                 "This is text editor.\n(test version)")

    def init_ui(self):
        ''' Create all controls '''
        self.parent.title("TkInter demo")
        self.pack(fill=BOTH, expand=True)

        title_frame = Frame(self)
        title_frame.pack(fill=X)

        title_label = Label(title_frame, text="Disc title", width=6)
        title_label.pack(side=LEFT, padx=5, pady=5)

        self.disc_entry = Entry(title_frame)
        self.disc_entry.pack(fill=X, padx=5, expand=True)

        options_frame = Frame(self)
        options_frame.pack(fill=X)

        self.check_cd = IntVar()
        self.check_dvd = IntVar()
        checkbutton_cd = Checkbutton(options_frame,
                                     text="CD",
                                     variable=self.check_cd)
        checkbutton_dvd = Checkbutton(options_frame,
                                      text="DVD",
                                      variable=self.check_dvd)
        checkbutton_cd.pack(side=LEFT, padx=5, pady=5)
        checkbutton_dvd.pack(side=LEFT, padx=5, pady=5)

        listbox_frame = Frame(self)
        listbox_frame.pack(fill=BOTH, expand=True)

        self.cd_listbox = Listbox(listbox_frame)
        self.cd_listbox.pack(fill=X)
        self.dvd_listbox = Listbox(listbox_frame)
        self.dvd_listbox.pack(fill=X)

        self.help_button = Button(listbox_frame, text="Turn
Red!")
        self.help_button.pack(side=RIGHT, anchor=N, padx=5,
pady=5)
        self.help_button['borderwidth'] = 8
        self.help_button['height'] = 30
        self.help_button.bind("<Button-1>", self.turn_red)

        self.add_button = Button(listbox_frame, text="Add disc")
        self.add_button.pack(side=RIGHT, anchor=N, padx=5,
pady=5)
        self.add_button['height'] = 30
        self.add_button.bind("<Button-1>", self.add_disc)


def main():
    root = Tk()
    # не нужно быть семи пядей во лбу,
    # дабы понять, что означают эти циферки:
    root.geometry("400x460+300+300")
    app = DiscFrame(root)
    # как всегда цикл обработки сообщений:
    root.mainloop()
```

```
if __name__ == '__main__':
    main()
```

wxPython:

```
# encoding: utf8
import wx

"""
В данном коде будут одновременно созданы три окна.

--- Просто фрагмент из книги о wxpython:

There are three ways to create the ID numbers used by a
widget:
1 Explicitly pass a positive integer into the constructor
2 Get wxPython to create IDs for you using the wx.NewId()
function
3 Pass either the global constant wx.ID_ANY or -1 to a widget
constructor
"""


class MainFrame(wx.Frame):
    ''' Класс главного окна '''

    def __init__(self):
        wx.Frame.__init__(self, None, -1, "Main Frame",
size=(500, 400))
        self.create_menu()

        panel = wx.Panel(self, -1)
        self.tree_ctrl = wx.TreeCtrl(panel, -1, size=(200, 320),
pos=(10, 10),
                                     style=wx.TR_DEFAULT_STYLE |
                                     wx.TR_FULL_ROW_HIGHLIGHT |
                                     wx.TR_EDIT_LABELS)

        self.list_box = wx.ListCtrl(panel, -1,
style=wx.LC_REPORT,
                                    size=(250, 200), pos=(230, 120))
        self.list_box.InsertColumn(0, "Title")
        self.list_box.InsertColumn(1, "Type")

        wx.StaticText(panel, -1, "CD title:", pos=(230, 20))
        self.text_cd = wx.TextCtrl(panel, -1, size=(190, 20),
pos=(290, 20))

        message_button = wx.Button(
            panel, -1, "Add!", size=(250, 30), pos=(230, 70))
        self.Bind(wx.EVT_BUTTON, self.on_add_disc,
message_button)

        self.statusbar = self.CreateStatusBar()
        self.statusbar.SetFieldsCount(3)
        self.statusbar.SetStatusWidths([-2, -2, -1])
        self.statusbar.SetStatusText("wxPython Program", 1)
        self.statusbar.SetStatusText("(c) 2016 Author", 2)

        self.init_model()

    def init_model(self):
        self.discs = {'cd': ['1972 - Foxtrot', '1980 - Duke', '2002 -
Up'],
                      'dvd': ['Forrest Gump', 'Home Alone']}

        self.update_views()

    def update_views(self):
        self.tree_ctrl.DeleteAllItems()
        self.list_box.DeleteAllItems()

        root = self.tree_ctrl.AddRoot('Disc Collection')

        for d in self.discs:
            disc_type = self.tree_ctrl.AppendItem(root, d)
            for item in self.discs[d]:
                self.tree_ctrl.AppendItem(disc_type, item)

        i = 0
        for d in self.discs:
            for item in self.discs[d]:
                self.list_box.InsertStringItem(i, item)
                self.list_box.SetStringItem(i, 1, d)
                if d == 'dvd':
                    self.list_box.SetItemBackgroundColour(i,
"lightgreen")
                else:
                    self.list_box.SetItemBackgroundColour(i,
"lightyellow")
                i += 1

    def menu_data(self):
        data = (("&Operations",
                 ({"&Add!": self.on_add_disc},)),
                ("&Help",
                 ({"&About...": self.on_about_click},
                  {"&Web-site": None})
                 ))
        return data

    def create_menu(self):
        menu = wx.MenuBar()

        for item in self.menu_data():
            menu_item = self.create_sub_menu(item[1])
            menu.Append(menu_item, item[0])

        self.SetMenuBar(menu)

    def create_sub_menu(self, itemgroup):
        groupmenu = wx.Menu()

        for item in itemgroup:
            # Каждый элемент меню представлен своим
названием
            # и (опционально) обработчиком на свое нажатие
            title, handler = item.items()[0]
            menu_item = groupmenu.Append(-1, title)
            if handler:
                self.Bind(wx.EVT_MENU, handler, menu_item)

        return groupmenu

    def on_add_disc(self, event):
        if self.text_cd.Value != "":
            self.discs['cd'].append(self.text_cd.Value)

        self.statusbar.SetStatusText("Disc was added!")
        self.update_views()

    def on_about_click(self, event):
        dlg = wx.MessageDialog(None, "This is the coolest thing
ever!",
                               "MessageDialog", wx.OK |
wx.ICON_INFORMATION)
        result = dlg.ShowModal()
```

```
        dlg.Destroy()


class StringFrame(wx.Frame):
    ''' Класс окна работы со строками '''

    def __init__(self):
        wx.Frame.__init__(
            self, None, -1, "Обработка строк", size=(400, 320))

        panel = wx.Panel(self, -1)
        wx.StaticText(panel, -1, "Строка:", pos=(5, 20))
        self.entry_text = wx.TextCtrl(
            panel, -1, "", size=(300, -1), pos=(70, 20))
        wx.StaticText(panel, -1, "Результат:", pos=(5, 230))
        self.result_text = wx.TextCtrl(
            panel, -1, "", size=(300, -1), pos=(70, 230))

        sc = wx.SpinCtrl(panel, -1, "", (295, 55), (40, -1))
        sc.SetRange(1, 20)
        sc.SetValue(5)

        wx.CheckBox(panel, -1, "Удалить слова размером
меньше",
                    (70, 60), (250, 20))
        wx.StaticText(panel, -1, "букв", pos=(340, 62))
        wx.CheckBox(panel, -1, "Заменить все цифры на *",
(70, 80), (220, 20))
        wx.CheckBox(panel, -1, "Вставить пробелы между
символами",
                    (70, 100), (280, 20))
        self.sort_checkbox = wx.CheckBox(panel, -1,
"Сортировать слова в строке",
                                         (70, 120), (220, 20))

        self.radio_by_size = wx.RadioButton(
            panel, -1, "По размеру", (100, 140), (150, 20))
        self.radio_by_lex = wx.RadioButton(
            panel, -1, "Лексикографически", (100, 160), (150, 20))
        self.radio_by_size.Disable()
        self.radio_by_lex.Disable()

        format_button = wx.Button(panel, -1,
"Форматирование",
                                  size=(300, 30), pos=(70, 190))

        self.Bind(wx.EVT_BUTTON, self.on_format_click,
format_button)
        self.Bind(wx.EVT_CHECKBOX, self.on_check,
self.sort_checkbox)

    def on_format_click(self, event):
        # кое-что реализуем...
        input = self.entry_text.Value
        output = ' '.join(input)
        self.result_text.Value = output

    def on_check(self, event):
        if self.sort_checkbox.IsChecked():
            self.radio_by_size.Enable()
            self.radio_by_lex.Enable()
        else:
            self.radio_by_size.Disable()
            self.radio_by_lex.Disable()


class LogFrame(wx.Frame):
    ''' Класс окна отображения лога '''

    def __init__(self):
        wx.Frame.__init__(
            self, None, -1, "Искатель строк", size=(500, 400))
        self.create_menu()
        self.statusbar = self.CreateStatusBar()
        self.statusbar.SetFieldsCount(2)
        self.statusbar.SetStatusWidths([-3, -2])

        sample_list = [r'Файл first.txt был обработан 05.03.2016
19:05:18:',
                       '',
                       'Строка 105, позиция 19 : найдено "5-12-
2011"',
                       'Строка 120, позиция 7 : найдено "22-10-
2012"',
                       '',
                       r'Файл example.txt был обработан 05.03.2015
19:08:24:',
                       '',
                       'Строка 3, позиция 10 : найдено "11-05-2014"',
                       'Строка 12, позиция 2 : найдено "23-11-2014"',
                       'Строка 12, позиция 17 : найдено "23-11-
2014"']
        list_box = wx.ListBox(self, -1, (20, 20), (80, 120),
                              sample_list, wx.LB_SINGLE)
        list_box.SetSelection(0)

        self.statusbar.SetStatusText(
            "Обработан файл example.txt")
        self.statusbar.SetStatusText("15 036 байт", 1)

    def menu_data(self):
        data = (("&Файл",
                 ({"&Открыть...": self.on_open_file},)),
                ("&Лог",
                 ({"&Сохранить новый...": self.on_save_log},
                  {"&Добавить в лог": None},
                  {"&Просмотр лога": None})
                 ))
        return data

    def create_menu(self):
        menu = wx.MenuBar()

        for item in self.menu_data():
            menu_item = self.create_sub_menu(item[1])
            menu.Append(menu_item, item[0])

        self.SetMenuBar(menu)

    def create_sub_menu(self, itemgroup):
        groupmenu = wx.Menu()

        for item in itemgroup:
            title, handler = item.items()[0]
            menu_item = groupmenu.Append(-1, title)
            if handler:
                self.Bind(wx.EVT_MENU, handler, menu_item)

        return groupmenu

    def on_open_file(self, event):
        dlg = wx.FileDialog(self, message="Выберите файл",
defaultDir="",
                            defaultFile="", wildcard="*.*",
style=wx.OPEN)

        # при открытии файла просто обновляем строку
состояния
        if dlg.ShowModal() == wx.ID_OK:
            self.statusbar.SetStatusText(dlg.GetPath())

    def on_save_log(self, event):
        # при попытке сохранения лога внезапно запускаем
окно ))
```

```
        dlg = StringFrame()
        dlg.Show()


if __name__ == '__main__':
    # создаем объект приложения
    app = wx.App()

    # создаем объект окна MainFrame
    main_frame = MainFrame()
    # и показываем
    main_frame.Show()

    # создаем объект окна MainFrame
    string_frame = StringFrame()
    # и показываем
    string_frame.Show()

    # создаем объект окна MainFrame
    log_frame = LogFrame()
    # и показываем
    log_frame.Show()

    # запускаем главный цикл обработки сообщений
    app.MainLoop()
```

PyQt:

1) В программе QtDesigner формируем необходимое окошко с лейаутом и контролами
2) Запускаем утилиту pyuic4 (на основе разметки *.ui будет сгенерирован модуль *.py)
3) Пишем код с обработчиками и главным циклом обработки сообщений:

In [1]:
```
import sys
from PyQt4.QtCore import *
from PyQt4.QtGui import *
from qt.test_ui import Ui_Dialog


class MyForm(QMainWindow):

    def __init__(self, parent=None):
        QWidget.__init__(self, parent)
        self.ui = Ui_Dialog()
        self.ui.setupUi(self)
        self.fill_list()

    def fill_list(self):
        words = ['Hello', 'man', 'How\'re', 'You', 'Doing?']
        m = QStandardItemModel(self)
        for word in words:
            item = QStandardItem(word)
            m.appendRow(item)

        self.ui.listView.setModel(m)

    def accept(self):

self.ui.listView.model().appendRow(QStandardItem('Yeah!'))

    def reject(self):
        self.close()


if __name__ == "__main__":
    app = QApplication(sys.argv)
    form = MyForm()
    form.show()
    try:
        sys.exit(app.exec_())
    except SystemExit as se:
        print('Program has exited with code {}'.format(se))
```

Лекция 10. Работа с базами данных в Python

- Выполнение SQL-запросов
- ORM на примере SQLAlchemy
- Немного о noSql на примере Mongo

In [1]:
```
import sqlite3
# import MySQLdb    - для MySQL
# import psycopg2   - для PostgreSQL
# import pymssql    - для MS SQL

# дальнейший cursor-based код одинаков для всех движков
```
In [2]:
```
DBNAME = r'db/catalog.db'


def create_database():
    # создаем базу данных sqlite
    # (присоединяемся к ней первый раз)
    db = sqlite3.connect(DBNAME)

    with db:
        # Получаем объект курсора
        cursor = db.cursor()

        # SQL-запрос #1:
        cursor.execute('''
            CREATE TABLE performers(id INTEGER PRIMARY
KEY,
                                    name TEXT,
                                    desc TEXT)
        ''')
        db.commit()

        # SQL-запрос #2:
        cursor.execute('''
            CREATE TABLE albums(
            id INTEGER PRIMARY KEY,
            name TEXT,
            release_year INTEGER,
            perfid INTEGER,
            FOREIGN KEY(perfid) REFERENCES
performers(id))
            ''')
        db.commit()


try:
    create_database()
    print('Database was created succesfully!')
except sqlite3.OperationalError as e:
    print(e)
```
```
Database was created succesfully!
```
```
def insert_records():
    # присоединяемся к базе данных sqlite
```

```
    db = sqlite3.connect(DBNAME)

    with db:
        cursor = db.cursor()

        ans = input('Do you wish to insert some performers?
(y/n)')
        while ans == 'y':
            perf = input('Performer: ')
            # Потенциально опасно!
            # НЕ ДЕЛАТЬ ТАК!
            cursor.execute('INSERT INTO performers
VALUES(NULL, "%s", "")' % perf)
            ans = input('Continue? (y/n)')

        db.commit()

        ans = input('Do you wish to insert some albums? (y/n)')
        while ans== 'y':
            perfID = input('Performer ID: ')
            albumname = input('Album name: ')
            albumyear = int(input('Album year: '))
            # Здесь лучше: делаем параметризированный запрос
            cursor.execute('INSERT INTO albums
VALUES(NULL, ?, ?, ?)',
                           (albumname, albumyear, perfID))
            ans = input('Continue? (y/n)')

        db.commit()


insert_records()
Do you wish to insert some performers? (y/n)y
Performer: 10cc
Continue? (y/n)y
Performer: Supertramp
Continue? (y/n)y
Performer: Renaissance
Continue? (y/n)n
Do you wish to insert some albums? (y/n)y
Performer ID: 1
Album name: Deceptive Bends
Album year: 1977
Continue? (y/n)y
Performer ID: 2
Album name: Breakfast in America
Album year: 1979
Continue? (y/n)y
Performer ID: 3
Album name: Novella
Album year: 1977
Continue? (y/n)n
In [4]:
def select_records():
    db = sqlite3.connect(DBNAME)

    with db:
        cursor = db.cursor()
        cursor.execute('SELECT id, name from performers')
        performers = cursor.fetchall()

        cursor.execute('''
                SELECT performers.name, albums.name,
albums.release_year
                from albums INNER JOIN performers ON
albums.perfID = performers.id
                ''')
        albums = cursor.fetchall()

        # fetchone():   считывает одну следующую строку
результирующего набора запроса.
        #               Результирующий набор - это объект,
возвращаемый
        #               объектом-курсором при выполнении запроса
к таблице;
        # fetchall():   считывает все строки в результирующем
наборе запроса.
        #               Если какие-то строки уже были извлечены
из результирующего набора,
        #               то метод возвращает оставшиеся строки
набора;
        # rowcount:     Это read-only атрибут, в котором
хранится количество строк,
        #               затронутых при вызове метода execute().

        return performers, albums


performers, albums = select_records()

print()
for performer in performers:
    print(performer)

print()
for album in albums:
    print(album)
(1, '10cc')
(2, 'Supertramp')
(3, 'Renaissance')

('10cc', 'Deceptive Bends', 1977)
('Supertramp', 'Breakfast in America', 1979)
('Renaissance', 'Novella', 1977)
ORM (Object-Relational Mapping)
In [5]:
import sqlalchemy
from sqlalchemy.orm import sessionmaker
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import Column, Integer, String, ForeignKey
from sqlalchemy.orm import relationship


Base = declarative_base()


class Performer(Base):
    __tablename__ = 'performers'

    id = Column(Integer, primary_key=True)
    name = Column(String)
    desc = Column(String)
    albums = relationship('Album', back_populates='performer')

    def __repr__(self):
        return "<Performer(name='%s', desc='%s')>" % (
                        self.name, self.desc)


class Album(Base):
    __tablename__ = 'albums'

    id = Column(Integer, primary_key=True)
    name = Column(String)
    release_year = Column(Integer)
    perfid = Column(Integer, ForeignKey('performers.id'))
    #performer = relationship('Performer', backref='albums')
    performer = relationship('Performer',
back_populates='albums')

    def __repr__(self):
        return "<Album(name='%s', year='%d')>" % (
                        self.name, self.release_year)
```

In [6]:
```
def insert_records_via_ORM():
    ans = input('Do you wish to insert some performers? (y/n)')
    while ans== 'y':
        name = input('Performer: ')
        performer = Performer(name=name, desc='')
        session.add(performer)
        session.commit()

        ans = input('Continue? (y/n)')

    ans = input('Do you wish to insert some albums? (y/n)')
    while ans == 'y':
        performer_name = input('Performer: ')
        performer = session.query(Performer) \
                    .filter(Performer.name == performer_name) \
                    .first()
        if performer:
            album_name = input('Album name: ')
            album_year = int(input('Album year: '))
            album = Album(name=album_name,
release_year=album_year)
            performer.albums.append(album)
            session.add(album)
            session.flush()
            session.commit()
        else:
            print('No such performer!')

        ans = input('Continue? (y/n)')
```

In [7]:
```
def select_records_via_ORM():
    performers = [performer.name
                  for performer in session.query(Performer)]
    albums = [(album.performer.name, album.name,
album.release_year)
              for album in session.query(Album)]
    return performers, albums
```

In [8]:
```
# Флаг echo включает ведение лога через стандартный
модуль logging Питона.
# Когда он включен, мы увидим все созданные нами SQL-
запросы.
engine = sqlalchemy.create_engine('sqlite:///' + DBNAME,
echo=False)
#engine = sqlalchemy.create_engine('sqlite:///:memory:',
echo=True)

Session = sessionmaker(bind=engine)
session = Session()

insert_records_via_ORM()

performers, albums = select_records_via_ORM()

print('\nPerformers:')
for performer in performers:
    print(performer)

print('\nAlbums:')
for album in albums:
    print(album)

session.close()
```

```
Do you wish to insert some performers? (y/n)y
Performer: Camel
Continue? (y/n)n
Do you wish to insert some albums? (y/n)y
Performer: Camel
Album name: A Nod And A Wink
Album year: 2002
Continue? (y/n)y
Performer: 10cc
Album name: Look Hear?
Album year: 1979
Continue? (y/n)y
Performer: Camel
Album name: Rain Dances
Album year: 1977
Continue? (y/n)n

Performers:
10cc
Supertramp
Renaissance
Camel

Albums:
('10cc', 'Deceptive Bends', 1977)
('Supertramp', 'Breakfast in America', 1979)
('Renaissance', 'Novella', 1977)
('Camel', 'A Nod And A Wink', 2002)
('10cc', 'Look Hear?', 1979)
('Camel', 'Rain Dances', 1977)
```

In [9]:
```
albums = [(album.performer.name, album.name,
album.release_year)
          for album in
session.query(Album).filter(Album.release_year == 1977)]
albums
```

Out[9]:
```
[('10cc', 'Deceptive Bends', 1977),
 ('Renaissance', 'Novella', 1977),
 ('Camel', 'Rain Dances', 1977)]
```

Лекция 11. Асинхронность и многопоточность

- Модуль threading
- Механизмы синхронизации
- Очереди, Producer/Consumer
- Глобальная блокировка интерпретатора (GIL)
- Асинхронность в Python
- Модуль asyncio

In [1]:
```
import threading
import time
```

In [2]:
```
def prime_numbers(lower, upper):
    ''' (неэффективная) функция для вывода простых чисел в
диапазоне '''
    if lower < 2:
        lower = 2
    for num in range(lower, upper + 1):
        for i in range(2, num):
            if num % i == 0:
                break
        else:
            print('- up to {} : pn={}'.format(upper, num))


t = threading.Thread(target=prime_numbers,
                     args=(2**20, 2**20 + 256))
t.start()
```

```
- up to 1048832 : pn=1048583
- up to 1048832 : pn=1048589
- up to 1048832 : pn=1048601
- up to 1048832 : pn=1048609
- up to 1048832 : pn=1048613
- up to 1048832 : pn=1048627
- up to 1048832 : pn=1048633
- up to 1048832 : pn=1048661
- up to 1048832 : pn=1048681
- up to 1048832 : pn=1048703
- up to 1048832 : pn=1048709
- up to 1048832 : pn=1048717
- up to 1048832 : pn=1048721
- up to 1048832 : pn=1048759
- up to 1048832 : pn=1048783
- up to 1048832 : pn=1048793
- up to 1048832 : pn=1048799
- up to 1048832 : pn=1048807
- up to 1048832 : pn=1048829
In [3]:
class PrimeNumberThread(threading.Thread):
    ''' Класс генерации простых чисел с функцией потока '''
    def __init__(self, lower, upper):
        self._lower = lower
        self._upper = upper
        threading.Thread.__init__(self)

    def run(self):
        for num in range(self._lower, self._upper + 1):
            for i in range(2, num):
                if num % i == 0:
                    break
            else:
                print('- up to {} : pn={}'.format(self._upper, num))


t = PrimeNumberThread(150, 200)
t.start()
- up to 200 : pn=151
- up to 200 : pn=157
- up to 200 : pn=163
- up to 200 : pn=167
- up to 200 : pn=173
- up to 200 : pn=179
- up to 200 : pn=181
```

Пример окна с индикацией прогресса (wxPython) wxPythonDemo

```
class DownloadThread(threading.Thread):
    ''' Класс простого потока индикации прогресса.

        _percentage - текущий процент выполнения операции
        _update - функция, которая вызывается
        _callback - функция, которая будет вызываться
    '''
    def __init__(self, thread_name, thread_func, callback):
        threading.Thread.__init__(self)
        self.daemon = True
        self.name = thread_name
        self._percentage = 0
        self._update = thread_func
        self._callback = callback

    def run(self):
        while self._percentage < 100:
            time.sleep(0.03)
            self._percentage += 1
            self._update(self._percentage)

        self._callback(self.name)
```

```
- up to 200 : pn=191
- up to 200 : pn=193
- up to 200 : pn=197
- up to 200 : pn=199
In [4]:
# пример одновременного выполнения 2 потоков:
t1 = PrimeNumberThread(2**20, 2**20 + 256)
t2 = PrimeNumberThread(2**21, 2**21 + 256)
t1.start()
t2.start()
- up to 1048832 : pn=1048583
- up to 1048832 : pn=1048589
- up to 2097408 : pn=2097169
- up to 1048832 : pn=1048601
- up to 1048832 : pn=1048609
- up to 2097408 : pn=2097211
- up to 1048832 : pn=1048613
- up to 1048832 : pn=1048627
- up to 2097408 : pn=2097223
- up to 1048832 : pn=1048633
- up to 1048832 : pn=1048661
- up to 2097408 : pn=2097229
- up to 1048832 : pn=1048681
- up to 1048832 : pn=1048703
- up to 2097408 : pn=2097257
- up to 1048832 : pn=1048709
- up to 1048832 : pn=1048717
- up to 2097408 : pn=2097259
- up to 1048832 : pn=1048721
- up to 1048832 : pn=1048759
- up to 2097408 : pn=2097287
- up to 1048832 : pn=1048783
- up to 2097408 : pn=2097289- up to 1048832 : pn=1048793

- up to 1048832 : pn=1048799
- up to 1048832 : pn=1048807- up to 2097408 : pn=2097311

- up to 1048832 : pn=1048829
- up to 2097408 : pn=2097317
- up to 2097408 : pn=2097349
- up to 2097408 : pn=2097373
- up to 2097408 : pn=2097383
- up to 2097408 : pn=2097397
- up to 2097408 : pn=2097401
```

```
class MainFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, wx.ID_ANY, 'Threads!',
size=(400, 300))

        panel = wx.Panel(self, -1)
        self.gauge1 = wx.Gauge(panel, -1, pos=(20,20),
size=(360,40))
        self.gauge2 = wx.Gauge(panel, -1, pos=(20,80),
size=(360,40))

        button_start1 = wx.Button(panel, -1, 'start thread #1',
pos=(20,140),
                                  size = (170, 50) )
        button_start2 = wx.Button(panel, -1, 'start thread #2',
pos=(210,140),
                                  size = (170, 50) )
        self.Bind(wx.EVT_BUTTON, self.on_start_thread1,
button_start1)
        self.Bind(wx.EVT_BUTTON, self.on_start_thread2,
button_start2)
```

```
        self.info_text = wx.StaticText(panel, -1, pos=(20, 230),
size=(360, 20))
        self.info_text.SetLabel('Idle...')

    def on_start_thread1(self,event):
        self.dthread1 = DownloadThread('1', self.update_gauge1,
self.download_finished)
        self.dthread1.start()

    def on_start_thread2(self,event):
        self.dthread2 = DownloadThread('2', self.update_gauge2,
self.download_finished)
        self.dthread2.start()

    def update_gauge1(self, percentage):
        self.gauge1.SetValue(percentage)

    def update_gauge2(self, percentage):
        self.gauge2.SetValue(percentage)

    def download_finished(self, thread_no):
        self.info_text.SetLabel('Thread {} has
finished'.format(thread_no))


if __name__ == '__main__':
    app = wx.App()
    frame = MainFrame()
    frame.Show()
    app.MainLoop()
```

Механизмы синхронизации потоков
мютексы (mutexes, lock)
семафоры (semaphores)
события (events)
conditions

События (events)

In [6]:

```
from random import randint

model = ''

def load_model():
    global model
    models = ['First model', 'Second model', 'Third model',
None]
    for m in models:
        views_updated.wait()
        views_updated.clear()
        model = m
        time.sleep(randint(1, 3))
        model_loaded.set()

def update_views():
    while True:
        model_loaded.wait()
        model_loaded.clear()
        if model is None:
            break
        for i in range(3):
            print('View {} has been updated: {}'.format(i+1,
model))
        views_updated.set()

# инициализируем события
model_loaded = threading.Event()
views_updated = threading.Event()

t1 = threading.Thread(target=load_model)
t2 = threading.Thread(target=update_views)
t1.start()
t2.start()

# устанавливаем "флажок" первого события
views_updated.set()

# присоединяем потоки к основному потоку
# (т.е. не двигаемся дальше, пока не завершатся потоки)
t1.join()
t2.join()
```

```
View 1 has been updated: First model
View 2 has been updated: First model
View 3 has been updated: First model
View 1 has been updated: Second model
View 2 has been updated: Second model
View 3 has been updated: Second model
View 1 has been updated: Third model
View 2 has been updated: Third model
View 3 has been updated: Third model
```

Механизмы блокировки

In [7]:

```
# Мютекс (lock)

lock = threading.Lock()

def count_numbers(n, delay):
    with lock:
        print('Printing numbers from 1 to', n)
        for i in range(1, n + 1):
            print(i)
            time.sleep(delay)

t1 = threading.Thread(target=count_numbers, args=(10, 1))
t2 = threading.Thread(target=count_numbers, args=(5, 2))

t1.start()
t2.start()

t1.join()
t2.join()
```

```
Printing numbers from 1 to 10
1
2
3
4
5
6
7
8
9
10
Printing numbers from 1 to 5
1
2
3
4
5
```

Producer/Consumer

In [8]:

```
# Работа с потоками через очередь Queue
# Класс Queue из модуля queue - потокобезопасная очередь
```

```
# (с помощью которой можно подавать данные с одного потока на другой).

from queue import Queue


# очередь
q = Queue()
# мютекс, блокирующий вывод на консоль в потоке
print_lock = threading.Lock()


def consumer():
    while True:
        # достаем из очереди задачу
        worker = q.get()
        # делаем что надо
        time.sleep(1)
        with print_lock:
            print('Поток {} обрабатывает задачу {}'.format(
                  threading.current_thread().name, worker))
        # отмечаем, что задача отработана
        q.task_done()

def producer(n):
    # каждый продюсер кладет в очередь 5 задач
    for task in range(5):
        q.put(n*5 + task)

# создаем 10 консьюмеров
for _ in range(10):
    t = threading.Thread(target=consumer, daemon=True)
    t.start()

# создаем 4 поставщиков
for i in range(4):
    t = threading.Thread(target=producer, daemon=True,
args=(i,))
    t.start()

# ждем завершения потока
q.join()
Поток Thread-25 обрабатывает задачу 9
Поток Thread-24 обрабатывает задачу 8
Поток Thread-22 обрабатывает задачу 6
Поток Thread-23 обрабатывает задачу 7
Поток Thread-21 обрабатывает задачу 5
Поток Thread-17 обрабатывает задачу 4
Поток Thread-20 обрабатывает задачу 3
Поток Thread-19 обрабатывает задачу 2
Поток Thread-18 обрабатывает задачу 1
Поток Thread-16 обрабатывает задачу 0
Поток Thread-18 обрабатывает задачу 18
Поток Thread-16 обрабатывает задачу 19
Поток Thread-19 обрабатывает задачу 17
Поток Thread-23 обрабатывает задачу 13
Поток Thread-22 обрабатывает задачу 12
Поток Thread-24 обрабатывает задачу 11
Поток Thread-25 обрабатывает задачу 10
Поток Thread-20 обрабатывает задачу 16
Поток Thread-17 обрабатывает задачу 15
Поток Thread-21 обрабатывает задачу 14
```

GIL (Global Interpreter Lock)

Если свести к одному предложению, то: в Python CPU-bound задачи, запрограммированные как многопоточные, по факту не многопоточны. Причиной этому - глобальная блокировка интерпретатора на уровне реализации CPython (семафоры ставятся практически на все участки сишного кода). Так уж сделал Гвидо, и у этого подхода есть серьезные преимущества (однопоточные скрипты работают более эффективно + дополнительная защита потоконебезопасных участков кода на C).
Подробно о внутренностях GIL пишет Девид Бизли:
http://www.dabeaz.com/python/GIL.pdf (старый GIL, < Python 3.2)
http://www.dabeaz.com/python/NewGIL.pdf (новый GIL)
Перевод на хабре:
https://habrahabr.ru/post/84629/
Из-за особенностей GIL + механизмов переключения потоков в любой ОС, код, "разнесенный" по потокам, будет работать даже медленнее, чем однопоточный.
Проверим это:

```
In [9]:
# Посчитаем сумму всех чисел из диапазона [1, 20000000)
# однопоточная версия
lower = 1
upper = 20000000

tm1 = time.time()

total = 0
for i in range(lower, upper):
    total += i

tm2 = time.time()

print('Single-threaded version: {} sec'.format(tm2 - tm1))
print('Result: {}'.format(total))
Single-threaded version: 1.9972162246704102 sec
Result: 199999990000000
In [10]:
# многопоточная версия

middle = 10000000

def count_sum(start, end, res):
    total = 0
    for i in range(start, end):
        total += i
```

```
    res[0] = total


sum1 = [0]
sum2 = [0]

tm1 = time.time()

t1 = threading.Thread(target=count_sum, args=(lower, middle, sum1))
t2 = threading.Thread(target=count_sum, args=(middle, upper, sum2))
t1.start()
t2.start()
t1.join()
t2.join()

total = sum1[0] + sum2[0]

tm2 = time.time()

print('Multi-threaded version: {} sec'.format(tm2 - tm1))
print('Result: {}'.format(total))
```
Multi-threaded version: 1.2708654403686523 sec
Result: 199999990000000
Согласно GIL, многопоточная версия не должна быть быстрее однопоточной. А у нас результат оказался противоположным. Причина - хитрая. В примере выше мы вызываем глобальную функцию, оперирующую глобальными переменными. Глобальные переменные хранятся в хеш-таблице. А потоки выполняются посредством своих функций со стеком переменных, и там (как и во всяких функциях) переменные хранятся в массивах. Скорость адресации в массиве выше скорости доступа к элементу хеш-таблицы. За счет этого получается выигрыш второй версии.
Ну а если напишем однопоточную версию с помощью функции, то получим ожидаемый результат:
In [11]:
```
lower = 1
upper = 20000000

tm1 = time.time()

total = [0]
count_sum(lower, upper, total)

tm2 = time.time()

print('Single-threaded version: {} sec'.format(tm2 - tm1))
print('Result: {}'.format(total[0]))
```
Single-threaded version: 1.2537884712219238 sec
Result: 199999990000000
NB. Время замерять часто удобнее с помощью timeit:
In [12]:
```
%timeit count_sum(lower, upper, total)
```
1 loop, best of 3: 1.13 s per loop
Асинхронность в Python
Асинхронность - то, без чего не обходится ни одна нормальная современная технология программирования. Многое из того, что в "лихие 90-ые" за неимением альтернатив делалось в потоках, сейчас лучше и эффективнее делать на асинхронных механизмах, а именно: всякие I/O задачи, где основной поток, выполняясь синхронно, тратит свое время на банальное ожидание ресурса, в то время как он мог бы преспокойно заниматься и другими делами. Примеры: загрузка файлов на локальный компьютер по сетевым протоколам; выполнение мудрёного для СУБД SQL-запроса и ожидание ответа; считывание данных с COM-порта с определенной периодичностью, и т.д.

Основные понятия асинхронного программирования:
event loops
awaitables
coroutines
generators
futures
concurrent futures
tasks
executors
transports
protocols

Все это есть в питончике. Читаем также здесь:
http://lucumr.pocoo.org/2016/10/30/i-dont-understand-asyncio/
Ниже рассмотрим примеры из официальной документации Python

In [13]:
```
import asyncio
import datetime
```
In [14]:

```
# Демонстрация event_loop
# call_soon, call_later

def display_date(end_time, loop):
    ''' показ времени каждую секунду '''
    print(datetime.datetime.now())
    if (loop.time() + 1.0) < end_time:
        loop.call_later(1, display_date, end_time, loop)
    else:
        loop.stop()

# Получаем себе цикл обработки сообщений (из текущего потока)
loop = asyncio.get_event_loop()

# вызываем функцию display_date()
end_time = loop.time() + 5.0
loop.call_soon(display_date, end_time, loop)

# говорим циклу выполняться вечно (но он прервется вызовом loop.stop() из функции)
loop.run_forever()
# Закрываем цикл обработки сообщений (в jupyter'e я это делать, конечно, не буду)
#loop.close()
2017-05-03 20:26:47.935598
2017-05-03 20:26:48.952208
2017-05-03 20:26:49.959222
2017-05-03 20:26:50.970285
2017-05-03 20:26:51.970992
In [15]:
@asyncio.coroutine
def display_date(loop):
    end_time = loop.time() + 5.0
    while True:
        print(datetime.datetime.now())
        if (loop.time() + 1.0) >= end_time:
            break
        yield from asyncio.sleep(1)

loop = asyncio.get_event_loop()
loop.run_until_complete(display_date(loop))
# Закрываем цикл обработки сообщений (в jupyter'e я это делать, конечно, не буду)
#loop.close()
2017-05-03 20:26:55.055535
2017-05-03 20:26:56.068854
2017-05-03 20:26:57.070970
2017-05-03 20:26:58.073680
2017-05-03 20:26:59.082370
Начиная с Python 3.5, такие конструкции "доведены до ума" и записываются более коротко:
asyncio.coroutine заменяется на async def foo()
yield from заменяется на await
In [16]:
async def display_date(loop):
    end_time = loop.time() + 5.0
    while True:
        print(datetime.datetime.now())
        if (loop.time() + 1.0) >= end_time:
            break
        await asyncio.sleep(1)

loop = asyncio.get_event_loop()
loop.run_until_complete(display_date(loop))
# Закрываем цикл обработки сообщений (в jupyter'e я это делать, конечно, не буду)
#loop.close()
2017-05-03 20:27:02.146636
2017-05-03 20:27:03.153434
2017-05-03 20:27:04.167310
2017-05-03 20:27:05.171045
2017-05-03 20:27:06.184588
In [17]:
"""
Две корутины в цепочке.

Цепочка: compute() -> print_sum().
Корутина print_sum() ожидает окончания compute() перед тем, как вернуть результат.
"""

async def compute(x, y):
    print("Compute %s + %s ..." % (x, y))
    await asyncio.sleep(1.0)
    return x + y

async def print_sum(x, y):
    result = await compute(x, y)
    print("%s + %s = %s" % (x, y, result))

loop = asyncio.get_event_loop()
loop.run_until_complete(print_sum(1, 2))
# цикл обработки сообщений (в jupyter'e я это делать, конечно, не буду)
# loop.close()
Compute 1 + 2 ...
1 + 2 = 3

In [18]:
# Параллельное выполнение трех тасков-задач (tasks),
# каждая из которых представлена корутиной вычисления факториала.

async def factorial(name, number):
    f = 1
    for i in range(2, number+1):
        print("Task %s: Compute factorial(%s)..." % (name, i))
        await asyncio.sleep(1)
        f *= i
    print("Task %s: factorial(%s) = %s" % (name, number, f))


loop = asyncio.get_event_loop()
loop.run_until_complete(asyncio.gather(
    factorial("A", 2),
    factorial("B", 3),
    factorial("C", 4),
))
# loop.close()
Task C: Compute factorial(2)...
Task B: Compute factorial(2)...
Task A: Compute factorial(2)...
Task C: Compute factorial(3)...
Task B: Compute factorial(3)...
Task A: factorial(2) = 2
Task C: Compute factorial(4)...
Task B: factorial(3) = 6
Task C: factorial(4) = 24
Out[18]:
[None, None, None]
In [19]:
# Пример создания тасков через create_task()

async def reorganize_data(filename):
    # просто пусть длительность паузы зависит от имени файла
    await asyncio.sleep(filename.index('.'))
    print('Reorganized data in file {}'.format(filename))


loop = asyncio.get_event_loop()

reorganize_task1 = loop.create_task(reorganize_data('12.csv'))
```

```
reorganize_task2 = loop.create_task(reorganize_data('3.csv'))

loop.run_until_complete(
    asyncio.gather(reorganize_task1, reorganize_task2))
# loop.close()
Reorganized data in file 3.csv
Reorganized data in file 12.csv
Out[19]:
[None, None]
In [20]:
# Пример работы с future; task - это подкласс future

async def read_file_contents(future):
    print('Ну, например, читаааааем из большого файла...
Ожидайте')
    print()
    # и здесь код чтения из файла, типа await
read_file_async()
    await asyncio.sleep(5)
    future.set_result('Прочитали:\nТекст файла... (и далее
миллион символов)')


future = asyncio.Future()

loop = asyncio.get_event_loop()
asyncio.ensure_future(read_file_contents(future))
loop.run_until_complete(future)

print(future.result())
#loop.close()
Ну, например, читаааааем из большого файла... Ожидайте

Прочитали:
Текст файла... (и далее миллион символов)
Ниже рассмотрим, как можно блокирующую функцию (не
asyncio-корутину) начать выполнять асинхронно:
In [20]:
import requests

async def download_urls():
    loop = asyncio.get_event_loop()
    future1 = loop.run_in_executor(None,
                                   requests.get,
                                   'https://docs.python.org/3/library/asyncio-
task.html')
    future2 = loop.run_in_executor(None,
                                   requests.get,
                                   'http://www.google.com')
    response1 = await future1
    response2 = await future2
    print('first site:')
    print(response1.text[:200])
    print()
    print('second site:')
    print(response2.text[:200])

loop = asyncio.get_event_loop()
loop.run_until_complete(download_urls())
# loop.close()
first site:
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">


<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv=

second site:
<!doctype html><html itemscope=""
itemtype="http://schema.org/WebPage"
lang="uk"><head><meta content="text/html; charset=UTF-8"
http-equiv="Content-Type"><meta
content="/images/branding/googleg/1x/goo
In [21]:
from concurrent.futures import ThreadPoolExecutor

# Пул потоков (executor) можем сами создать/настроить
executor = ThreadPoolExecutor(max_workers=3)

async def download_urls(executor):
    loop = asyncio.get_event_loop()
    future1 = loop.run_in_executor(executor,
                                   requests.get,
                                   'https://docs.python.org/3/library/asyncio-
task.html')
    future2 = loop.run_in_executor(executor,
                                   requests.get,
                                   'http://www.google.com')
    response1 = await future1
    response2 = await future2
    print('first site:')
    print(response1.text[:200])
    print()
    print('second site:')
    print(response2.text[:200])


loop = asyncio.get_event_loop()
loop.run_until_complete(download_urls(executor))
# loop.close()
first site:
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">


<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv=

second site:
<!doctype html><html itemscope=""
itemtype="http://schema.org/WebPage"
lang="uk"><head><meta content="text/html; charset=UTF-8"
http-equiv="Content-Type"><meta
content="/images/branding/googleg/1x/goo
In [22]:
# Исключения надо ловить обязательно самостоятельно,
# т.к. уведомлений о них никаких не будет
# (разве что если явно спросить у future)

async def suspicious():
    try:
        await some_buggy_operation()
    except ValueError as e:
        print(e)

async def some_buggy_operation():
    raise ValueError('Wow! Exception\n')


loop = asyncio.get_event_loop()
loop.run_until_complete(suspicious())
# loop.close()

print('Is loop running?', loop.is_running())
Wow! Exception

Is loop running? False
```

In [23]:
```
# корутины можно прерывать (cancellation):

async def poll(incr=1):
    """ бесконечная корутина """
    i = 0
    while True:
        print("Получены новые данные на {}
секунде".format(i))
        i += incr
        await asyncio.sleep(incr)


async def stop(duration):
    """ Корутина, ожидающая сигнал для остановки цикла
обрабокти сообщений """
    await asyncio.sleep(duration)
    # Также можно поднять здесь Exception и прописать
    # `return_when=asyncio.FIRST_EXCEPTION` в
`asyncio.wait`.


loop = asyncio.get_event_loop()
tasks = [asyncio.ensure_future(poll(2)),
         asyncio.ensure_future(poll(1.5)),
         asyncio.ensure_future(stop(5))]
finished, pending = loop.run_until_complete(
    asyncio.wait(tasks,
return_when=asyncio.FIRST_COMPLETED))

# отменяем остающиеся таски
for task in pending:
    print("Cancelling %s: %s" % (task, task.cancel()))
try:
    loop.run_until_complete(asyncio.gather(*pending))
except asyncio.CancelledError:
    for task in pending:
        print("Cancelled %s: %s" % (task, task.cancelled()))
```
```
Получены новые данные на 0 секунде
Получены новые данные на 0 секунде
Получены новые данные на 1.5 секунде
Получены новые данные на 2 секунде
Получены новые данные на 3.0 секунде
Получены новые данные на 4 секунде
Получены новые данные на 4.5 секунде
Cancelling <Task pending coro=<poll() running at <ipython-
input-23-46203e8d92bc>:10> wait_for=<Future cancelled>>:
True
Cancelling <Task pending coro=<poll() running at <ipython-
input-23-46203e8d92bc>:10> wait_for=<Future cancelled>>:
True
Cancelled <Task cancelled coro=<poll() done, defined at
<ipython-input-23-46203e8d92bc>:4>>: True
Cancelled <Task cancelled coro=<poll() done, defined at
<ipython-input-23-46203e8d92bc>:4>>: True
```

In [24]:
```
# пример асинхронного варианта паттерна
Producer/Consumer

import random

q = asyncio.Queue()


async def producer(num):
    for _ in range(random.randint(1, 5)):
        await q.put(num)
        await asyncio.sleep(random.random())

async def consumer(num):
    while True:
        value = await q.get()
        print('Принял приемщик №{} : значение
{}'.format(num, value))


loop = asyncio.get_event_loop()

producers = [loop.create_task(producer(i)) for i in range(7)]
consumers = [loop.create_task(consumer(i)) for i in range(5)]

loop.run_until_complete(asyncio.wait(producers))

for c in consumers:
    c.set_result(True)
```
```
Принял приемщик №0 : значение 0
Принял приемщик №0 : значение 1
Принял приемщик №0 : значение 2
Принял приемщик №0 : значение 3
Принял приемщик №0 : значение 4
Принял приемщик №0 : значение 5
Принял приемщик №0 : значение 6
Принял приемщик №0 : значение 6
Принял приемщик №1 : значение 5
Принял приемщик №2 : значение 0
Принял приемщик №3 : значение 3
Принял приемщик №3 : значение 6
Принял приемщик №0 : значение 5
Принял приемщик №1 : значение 1
Принял приемщик №2 : значение 4
Принял приемщик №3 : значение 1
Принял приемщик №4 : значение 1
Принял приемщик №0 : значение 1
Принял приемщик №1 : значение 3
Принял приемщик №2 : значение 4
Принял приемщик №2 : значение 3
Принял приемщик №4 : значение 4
Принял приемщик №0 : значение 3
```