**2Слайд. Ruby**

**Ruby** - динамический, рефлективный, интерпретируемый высокоуровневый язык программирования для быстрого и удобного объектно-ориентированного программирования. Язык обладает независимой от операционной системы реализацией многопоточности, строгой динамической типизацией, сборщиком мусора и многими другими возможностями. По особенностям синтаксиса он близок к языкам Perl и Eiffel, по объектно-ориентированному подходу — к Smalltalk. Также некоторые черты языка взяты из Python, Lisp, Dylan и Клу.

Создатель Ruby — Юкихиро Мацумото (Matz) — интересовался языками программирования, ещё будучи студентом, но идея о разработке нового языка появилась позже. Ruby начал разрабатываться 23 февраля 1993 года и вышел в свет в 1995 году.

**3Слайд.**

Целью разработки было создание «настоящего объектно-ориентированного», лёгкого в разработке, интерпретируемого языка программирования.

**4Слайд.Семантика**

Ruby — полностью объектно-ориентированный язык. В нём все данные являются объектами, в отличие от многих других языков, где существуют примитивные типы. Каждая функция — метод. Он использует вызов по ссылке, то есть то есть при изменении значения переменной *a*, неявно изменилось и значение *b*, так как они содержат ссылку на один объект. То есть механизм присваивания действует одинаково для всех объектов.

**5Cлайд.**

Ruby – не поддерживает множественное наследование, но зато у него есть механизм примесей (mixis).( Это происходит следующим образом: в модуле определяются методы, потом определенный класс подключает этот модуль - и объекты класса получают все методы, определенные в модуле.) Так как Ruby мультипарадигменным языком: он поддерживает процедурный стиль, (определение функций и переменных вне классов), объектно-ориентированный (всё — объект), функциональный (анонимные функции, замыкания, возврат значения всеми инструкциями, возврат функцией последнего вычисленного значения). Он поддерживает отражение, метапрограммирование, информацию о типах переменных на стадии выполнения.

**6Слайд. Основной синтаксис**

На слайде мы видим небольшой отрывок кода на Руби:

**Классы. Объявление и наследование.**

**class** – ключевое слово для объявление класса. Символ **<** используется для наследования. Объявление в класса заканчивается ключевым словом **end**, как любой другой блок кода.

Доступ к родительскому классу осуществляется с помощью ключевого слова **super**.

**7Слайд. Основной синтаксис**

**Конструктор, методы и члены класса.**

Объявление метода в классе начинается с ключевого слова **def**, затем следует имя метода, и параметры.

Метод-конструктор класса должен называться **initialize**.

Любая переменная, имя которой начинающаяся с одного символа **@** - член класса. С двух символов **@** - статический член класса.

**8Слайд. Основной синтаксис**

**Вызов метода.**

Метод вызывается, как и в большинстве языков, через точку, скобки с перечислением параметров после вызова можно опустить, если это не вызывает недоразумений.

**9Слайд. Основной синтаксис**

**Ruby-символы.**

Идентификатор, и символ двоеточия в начале – это специальный объект в ruby, ruby-символ.

В большинстве случаев можно считать, что это ссылка на строку. Точнее, что-то, что представляет строку или имя.

Два ruby-символа с одинаковым именем – это один и тот же объект.

**10Слайд. Основной синтаксис**

Весь код в объявлении класса начинает выполнятся как только интерпретатор видит его.

**attr\_reader** и **attr\_accessor** – вызов метода класса Module, добавляющего в класс методы для чтения и доступа к членам класса **@name** и **@rocket** соответственно.

**11Cлайд. Основной синтаксис**

Классы в ruby открыты и свободны для дополнения.

Здесь в стандартный класс **Array** добавляется собственный метод, который будет виден только на время действия этого кода.

**12Слайд. Основной синтаксис**

**Ruby-блоки.**

Ruby-блоки – специальная конструкция языка. Код, объявленный внутри **do**..**end** выполняется внутри метода, с которым используется блок.

**select** и **each** – стандартные методы для выбора по условию и перебора всей коллекции соответственно.

**13Слайд. Основной синтаксис**

Код, написанный внутри блока, выполняется внутри метода, принимающего его. Этот метод рассматривает блок, как функцию, в которую можно передать какие-то параметры.

Вызов блока и передача параметров происходит с помощью ключевого слова **yield**.

**14Слайд. Разделители выражений (комментарий и перенос)**

**15Слайд. Блоки и замыкания**

Что такое блок в Ruby?

Блок — это произвольный код, который можно передать *любому* методу в качестве неявного последнего аргумента. Следует понимать, что при этом блок является особой конструкцией языка и обособлен от списка явных аргументов метода, что означает следующее:

* методу можно передать только один блок;
* он всегда идет в самом конце и вынесен за скобки с аргументами (если они есть);
* его можно задать при вызове любого метода, независимо от того, указаны в определении метода аргументы или нет.

Код может находиться внутри фигурных скобок { } или ключевых слов do end. Что именно использовать — решать вам, потому что обе конструкции описывают совершенно одинаковые по свойствам блоки. Обычно в фигурные скобки заключают код, состоящий из одной строки, при этом метод и его блок записывают на одной строке. Если код блока слишком длинный или объемный, вместо скобок на строке с вызовом метода оставляют do, а сам код пишут с новой строки с отступом и в конце «закрывают» его end'ом.

**16Слайд. Определение и вызов блока**

yield — это часть синтаксиса языка, как и объявление блока (поэтому ее нельзя переопределить). Когда в теле метода мы вызываем yield, то подразумеваем следующее: «отдай управление блоку, а когда тот закончит, верни управление этому методу».

**Лямбды**–анонимные функции **Синтаксис**:

lambda{|переданные переменные| код}

Лямбды похожи на процедуры, но есть два отличия:  
    1. Лямбды проверяют количество переданных аргументов.  
    2. Возврат из процедуры прекратит выполнение метода  и вернет какой - либо результат,                 люмбды возвращают результат в метод не прерывая его выполнения.

В proc\_return  выполнение останавливается и возвращается строка "Proc.new"  
Метод lambda\_return содержит  содержит анонимную функцию, которая возвращает строку "lambda", продолжая выполнение, доходит до следующего return и печатает строку "lambda\_return method finished"   
Процедуры в Ruby являются определенными фрагментами кода, но не методами. Поэтому return в процедуре является return в proc\_retur. Лямбды работают  как метод, который может иметь литерал return, не нарушая данное требование.

**17Слайд. Прокси класс**

Ruby является полностью объектно-ориентированным языком: числа, строки, регулярные выражения, массивы - это все объекты определенных классов. Класс определяет поведение объекта - он содержит все методы, доступные его объектам (инстансам).

При включении класса руби создает анонимный класс, и помещает его в цепочку наследования прямо над включившим этот модуль классом. Такие анонимные классы часто называют прокси-классами. Соответственно, при поиске метода, определенного в модуле, он будет найден в анонимном прокси-классе, и все произойдет так, как если бы метод был определен в одном из настоящих классов.

**18Слайд. Синглтон-классы**

В ruby есть специальная форма определения метода: def object.method\_name. Созданный таким образом метод называется синглтон-методом - он определен только для этого конкретного объекта.

Именно благодаря этой особенности в ruby работают классовые методы: каждый класс является объектом класса Class, который предоставляет лишь необходимый минимум методов - new, superclass и т.д.

**19Слайд. Self и текущий класс**

В каждой точке кода определены так называемые “текущий объект” и “текущий класс”.

Текущий объект - это объект, который доступен через псевдопеременную self; к нему адресуются все вызовы методов без указания получателя и в нем ищутся все запрошенные инстанс-переменные.

Текущий класс - это тот класс, инстанс-методом которого становится определенный в этом месте метод.

К сожалению, в отличие от текущего объекта, который можно узнать через self, текущий класс можно отслеживать только по коду.

**20Слайд. Верхний уровень**

До того, как мы входим в определение какого-либо класса, self указывает на main, а текущим классом является Object. Это объясняет тот факт, что методы, определяемые на верхнем уровне, вызываются из любого участка кода, причем без получателя (такой метод будет приватным, т.е. его нельзя вызывать с явным получателем; а self указывает на объект класса Object либо его потомка, поэтому метод будет доступен).

**21Слайд. Сравнение с Java**

\* Ruby -- интерпретируемый язык;

\* в Ruby все является объектом (в Java есть типы int и Integer, что создает определенные неудобства);

\* переменные в Ruby не являются статически типизированными и не требуют объявления;

\* модули (modules) в Ruby позволяют с помощью <<миксинов>> (mixins) конструировать подобие интерфейсов (interfaces) языка Java, допуская при этом в них реализацию методов.

**22Слайд. Заключение**

Этот язык, несомненно, является одним из лучших в качестве первого языка программирования. Быстрый цикл разработки (редактирование - запуск - редактирование), использование интерпретатора, изначальная объектно-ориентированность, нетипизированные переменные, которые не требуют объявления, - все это позволяет учащимся сконцентрировать свое внимание на общих принципах программирования.

Не менее важны мультиплатформенность Ruby и его принадлежность к миру свободно распространяемого ПО.

Еще одним важным фактором является простота методов, используемых в Ruby для ввода данных с клавиатуры или из файла. Вообще, основным аргументом в пользу Ruby в качестве первого языка программирования является именно его интуитивная ясность и предсказуемость, а не многие другие его преимущества.