**CORE JAVA Syllabus:**

**1. Core Java Programming Introduction of Java**

Introduction to Java; features of Java

Comparison with C and C++

Download and install JDK/JRE (Environment variables set up)

The JDK Directory Structure

First Java Program through command prompt

First Java Program through Eclipse

**2. Data types and Operators**

Primitive Datatypes, Declarations, Ranges

Variable Names Conventions

Numeric Literals, Character Literals

String Literals

Arrays(One dimensional; two- dimensional)

Array of Object References

Accessing arrays, manipulating arrays

Enumerated Data Types

Non-Primitive Datatypes

Defining a class, variable and method in Java

Method Signature; method calls

Expressions in Java; introduction to various operators

Assignment Operator

Arithmetic Operators

Relational Operators

Logical Operators

Conditional Operators

Operator Precedence

Implicit Type Conversions

Upcasting and downcasting

Strict typing

Type conversion

**3. Control Flow statements**

Statements and it's various categories in Java

if, if-else, if-else-if, switch case

for statement (both flavours traditional and enhanced for)

while and do-while loops

The continue Statement; labelled continue statement

The break Statement; labelled break statement

* Labelled statement is able to jump to the outer loop

return statement

**4. OOPS and its application in Java**

Classes and Objects: object is an instance of a class

Defining a class;Defining instance variables and methods

Creating objects out of a class

Method calls via object references

Abstraction:

* Hide implementation details
* May or may not contain abstract methods
* One abstract method 🡪 must be abstract class
* Abstract class cannot be instantiated
* Inherit and then implement
* All methods must be implemented

Interfaces and Abstract classes:

* You can extend multiple interface classes
* All methods in interface are final static public
* You can have public, private, protected variables etc.

Abstract and non-abstract methods

Inheritance

extends and implements keywords in Java

Super class and Sub class

this keyword, super keyword in Java for inheritance

Concrete classes in Java:

* **Concrete class in Java** is the default **class** and is a derived **class** that provides the basic implementations for all of the methods that are not already implemented in the base **class**. The opposite of the **concrete class** is the abstract **class**, which does not provide implementations for all of its methods.

Polymorphism

Compile time polymorphism -- Overloading of methods

Run time polymorphism -- Overriding of methods

Method  Overriding rules and method overloading rules

Introduction to Object class and it's methods

Encapsulation

Protection of data

Java Bean, POJO

Getters/Setters

Memory management in Java

Heap

Stack

**5. Packages**

Need for packages

What are packages; package declaration in Java

Import statement in Java

How do packages resolve name clashes?

**6. Miscellaneous**

Var-Args

Reference variables, local variables, instance variables

Memory allocations to variables

Double equals operator(==) operator for primitives and objects

toString() method on an object

**7. Statics**

Static variables and methods: created when program starts, destroyed when program ends

Static imports: less coding, e.g. import static java.lang.System.out println(“hi”)

Static initialization blocks; instance intialization blocks

Static concept in inheritance

**8. Constructors**

What are Constructors?

Properties of Constructors

Default and Parameterized Constructors

Rules for constructor implementation

Constructor Chaining

this call; super call for constructors

Constructors for Enumerated Data Types

Constructors concept for Abstract classes and interfaces

**9. Exceptions in Java**

What are Exceptions?

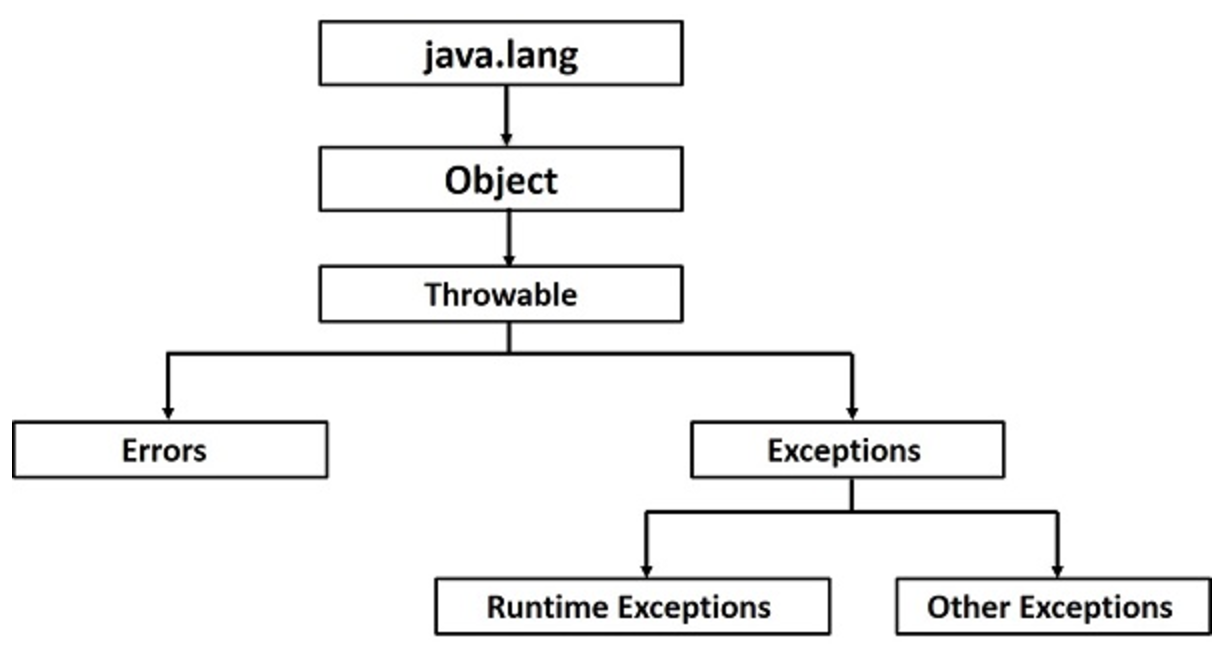
Need for exceptions

How can Exceptions be coded in Java?

API heirarchy for Exceptions

Types of Exceptions:

* **Checked Exceptions**: Checked exceptions − A checked exception is an exception that occurs at the compile time, these are also called as compile time exceptions. These exceptions cannot simply be ignored at the time of compilation, the programmer should take care of (handle) these exceptions.
* **Unchecked exceptions** − An unchecked exception is an exception that occurs at the time of execution. These are also called as Runtime Exceptions. These include programming bugs, such as logic errors or improper use of an API. Runtime exceptions are ignored at the time of compilation.
* Errors: e.g. stackoverflow



* **Casting**: ClassCastException
* **Arrays**: ArrayIndexOutOfBoundsException, NullPointerException
* **Collections**: NullPointerException, ClassCastException (if you're not using autoboxing and you screw it up)
* **IO**: java.io.IOException, java.io.FileNotFoundException, java.io.EOFException
* **Serialization**: java.io.ObjectStreamException (AND ITS SUBCLASSES, which I'm too lazy to enumerate)
* **Threads**: InterruptedException, SecurityException, IllegalThreadStateException
* **Potentially common to all situations**: NullPointerException, IllegalArgumentException

Keywords in Exception API: try, catch, finally, throw, throws

Rules for coding Exceptions

Declaring Exceptions

Defining and Throwing Exceptions

Errors and Runtime Exceptions

Custom Exception

Assertions

What are Assertions?

Enabling and disabling assertions in development environment

**10. Strings in Java**

What are Strings?

String heap memory and Constant Pool memory

Immutability in Strings

String creation on heap and constant pool

Method APIs on String; operations on Strings

Mutability of String Objects - StringBuilder and StringBuffer

Splitting of Strings and StringTokenizer class

**11. Collection Framework in Java**

The Collections Framework

The Set Interface

Set Implementation Classes

The List Interface

List Implementation Classes

The Map Interface

Map Implementation Classes

Queue Interface

Queue Implmentation classes

Utility classes

Sorting collections using utility methods

equals() and hashCode contract in Java collections

overriding equals and hashCode methods in Java

New Collections added in Java 1.6

Primitive wrapper classes and all its method APIs

**12. Generics**

Generics for Collections

Generics for class

Generics for methods

**13. Input-Output in Java**

What is a stream?

Overview of Streams

Bytes vs. Characters

Overview of the entire Java IO API

Reading a file; writing to a file usinf various APIs

Reading User input from console

PrintWriter Class

**14. Serialization**

Object Serialization

Serializable Interface

Serialization API

ObjectInputStream and ObjectOutput

Transient Fields

readObject and writeObject

**15. Inner Classes**

Inner Classes

Member Classes

Local Classes

Anonymous Classes

Static Nested Classes

**16. Threads in Java**

Non-Threaded Applications

Threaded Applications

Process based multitasking Vs Thread based multitasking

Thread API in Java

Creating Threads

States of a Thread

Sychronization for threads; static and non-static synchronized methods; blocks; concept of object and class locks

Coordination between threads - wait, notify and notifyAll methods for inter-thread communication

**17. Applets**

What are applets?

Need for Applets

Different ways of running an applet program

Applet API heirarchy

Life Cycle of an applet

Even Handlers for applets, mouse events, click events

**18. Swing GUI**

Introduction to AWT

Introduction to Swing GUI

Advantages of Swing over AWT

Swing API

Swing GUI Components

Event Handlers for Swing

Sample Calculator application using Swing GUI and Swing Event handling

**19. JDBC**

What is JDBC; introduction

JDBC features

JDBC Drivers

Setting up a database annd cretaing a schema

Writing JDBC code to connect to DB

CRUD Operations with JDBC

Statement types in JDBC

Types of Rowset, ResultSet in JDBC

**20. Access Modifers in Java**

What are access modifiers?

Default

Protected

Private

Public

**21. Debugging of Java Programs in Eclipse.**

**ADVANCED JAVA Syllabus:**

**Servlets and JSPs**

**1. Basics of a Web application**

What is a web application?

What is a web client and web server?

How do client and server communicate?

HTTP protocol basics

HTML language basics

What is a TCP/IP port, URL?

Need for a Web Container

**2. Web Contaner and Web Application Project Set up**

To set up Tomcat Container on a machine

To set up a Servlets JSP project in Eclipse

To configure dependency of Servlet JSP APIs

Web application project struture

**3. Servlets**

What are Servlets?

What can they do? Why are they needed?

How do Servlets look in code?

HTTP Methods; GET, POST, PUT, DELETE, TRACE, OPTIONS

GET/POST request; differences between the two

Servlet Lifecycle

Servlet Context and Servlet Config

Forwarding and Redirection of requests

**4. Session Management**

What is a session?

Why is it required?

How to get a session?

Session information passing between client and server

Sesssion information passing mechanisms - Cookies, Rewriting

How to destroy a session

**5. JSPs**

Introduction to JSP an dneed for JSPs

Basic HTML tags

JSP Lifecycle

**6. JSP Elements**

Scriptlets

Expressions

Declarations

Significance of above elements and fitment into the JSP Lifecycle

What are Directives in JSP?

Page Directive

Include Driective

Taglib Directive

**7. JSP Tag library**

JSP Standard Actions

Expression Language

JSTL basics and it's usage

Need for Custom Tag Library

Custom Tag Library implementation

**Struts Framework (version 2.x)**

**1. Basics of MVC**

What is MVC?

MVC Type1 and Type2 architecture

Why Struts framework?

Struts 1 overview

Struts 1 and Struts 2 comparison

**2. Struts 2 Architectutre**

Architecture Diagram explanation of following components:

Components of Model, Views and Controller in Struts Framework

Interceptors

Model/Action classes

Value Stack

OGNL

Introduction to configurations; framework and application architecture

Declarative and Annotations configuration approaches

**3. Struts 2 set up and first Action class**

Download JAR files

Struts 2 project build up and Configuration files

To build Action class

To intercept an HTTP request via Struts2 framework using Action class

Defining data and business logic in Action class

Preparing and Forwarding control to Views

**4. Struts 2 Interceptors**

What are Interceptors

Responsibilities of an Interceptor

Mechanism of Interceptor calling in Struts 2

Defining Interceptors

Defining Interceptor stacks

Defining Custom Interceptors

**5. Struts 2 Tag Library**

Introduction to tag library of Struts 2 and it's usage

**6. Struts 2 Validations**

Validations using Validateable interface

Workflow interceptor mechanism for validations

Validations using Validateable interface

Validation Framework introduction and architecture

Validating user input with above two mechanisms

**7. Struts 2 Tiles Frameworks**

Introduction to Tiles in a page

Struts2 Tiles framework introduction

Defining tiles.xml file

Configuring pages for tiles

A complete Tiles example with Struts2

**Hibernate Framework (version 3.x)**

**1. Introduction**

What is ORM principle?

Why ORM?

ORM implementations

**2. Hibernate Architecture**

Introduction to Hibernate

Hibernate Architecture

What are Peristent classes?

**3. Hibernate CRUD**

Setting up Hibernate project

Configuring all JARs and XML files

Setting up connection to DB using Hibernate

Performing basic CRUD operations using Hibernate API

Object Identity; Generator type classes

Using SQL with Hibernate

Using HQL

Using Criteria queries

**4. Mapping Collections and Associations**

To define sets, mas, lists in Hibernate

Association Mappings: 1. One to one

2 One to many

3 Many to one

4 Many to many

**5. Hibernate Caching**

What is caching?

What are the types of caching in Hibernate?

Explanation of various caching mechanisms in Hibernate

**6. Using Hibernate Annotations (if time permits)**

Sample example of using Hibernate Annotations

**Spring Framework (version 3.x)**

**1. Introduction to spring**

What is Spring?

Spring Architecture explanation and all it's components

**2. Introduction to all modules of spring**

Spring Bean Factory

Spring Application Context

Spring DI

Spring AOP

Spring Integration; Spring messaging, Spring JMS

Spring MVC

Spring DAO

**3. Setting up spring**

Setting up of Spring framework

Download JARs

Configure XML files

**4. Dependency Injection**

What is Dependency Injection?

How is it implemented using Spring Framework?

Bean Wiring mechanisms in Spring

**5. Spring AOP**

What is Spring AOP?

Implementation of Spring AOP

Algorithm:

interview questions:

algo:

# **HashSet VS TreeSet:**

## [**HashSet**](http://docs.oracle.com/javase/8/docs/api/java/util/HashSet.html)

* class offers constant time performance for the basic operations (add, remove, contains and size).
* it does not guarantee that the order of elements will remain constant over time
* iteration performance depends on the initial capacity and the load factor of the HashSet.
  + It's quite safe to accept default load factor but you may want to specify an initial capacity that's about twice the size to which you expect the set to grow.

## [**TreeSet**](http://docs.oracle.com/javase/8/docs/api/java/util/TreeSet.html)

* guarantees log(n) time cost for the basic operations (add, remove and contains)
* guarantees that elements of set will be sorted (ascending, natural, or the one specified by you via its constructor) (implements [SortedSet](http://docs.oracle.com/javase/8/docs/api/java/util/SortedSet.html))
* doesn't offer any tuning parameters for iteration performance
* offers a few handy methods to deal with the ordered set like [first()](http://docs.oracle.com/javase/8/docs/api/java/util/TreeSet.html#first--), last(), [headSet()](http://docs.oracle.com/javase/8/docs/api/java/util/TreeSet.html#headSet-E-), and [tailSet()](http://docs.oracle.com/javase/8/docs/api/java/util/TreeSet.html#tailSet-E-) etc
* **LinkedHashSet** is in some sense intermediate between HashSet and TreeSet. Implemented as a hash table with a linked list running through it, however **it provides insertion-ordered iteration which is not same as sorted traversal guaranteed by TreeSet**.

find islands in see

DP: max wealth to steal

trace multi-thread to find a value

BST

java and spring:

----------> https://www.tutorialspoint.com/java/

Comparable vs Comparator

hashmap vs hashtable

spring design pattern

spring implements hibernate how:

singleton design pattern

Spring dependency injection?

features and architectural design of the current system

caching framework, how to implement

spring singleton vs java singleton

download multiple files

Questions:

1. Right view of a binary search tree

2. Java threads questions, Build your own concurrent hash map

3. struts framework

4. how to make a class immutable

5. How set will identify if duplicate record has been inserted

6. Explain Map functionality

7. Tell about concurrency program in Java

8. OOPs concepts

9. Convert from tree to array

multithreading interview:

http://javarevisited.blogspot.sg/2014/07/top-50-java-multithreading-interview-questions-answers.html

map: mapping List<Person> to List<Integer>, integer refers to maybe the age of the person

filter: List<Integer> to List<Integer>, subset

reduce: similar to SQL aggregation

lambda expression: similar to Python

new type of object for lambda expression

lambda can be passed as params

Stream: typed interface

- not a collection

- a way to process large amount of data

- parallel --> multicore cpus

- Pipelined, avoid unnecessary intermediary computations

- an object on which to define operations

- not holding any data

- should not change data in processes as a good practice

- only to distribute the data to different paralleled processes

- can do data filtering :-), but the stream does not contain any data

- Intermediary action vs Final action, peek is intermediary, forEach is final

- anything that returns a stream is intermediary

go through the system you are working on right now

Machine Learning:

Supervised learning (dataset with which right answers are given), unsupervised learning (dataset for which right answers are not given), reinforcement learning

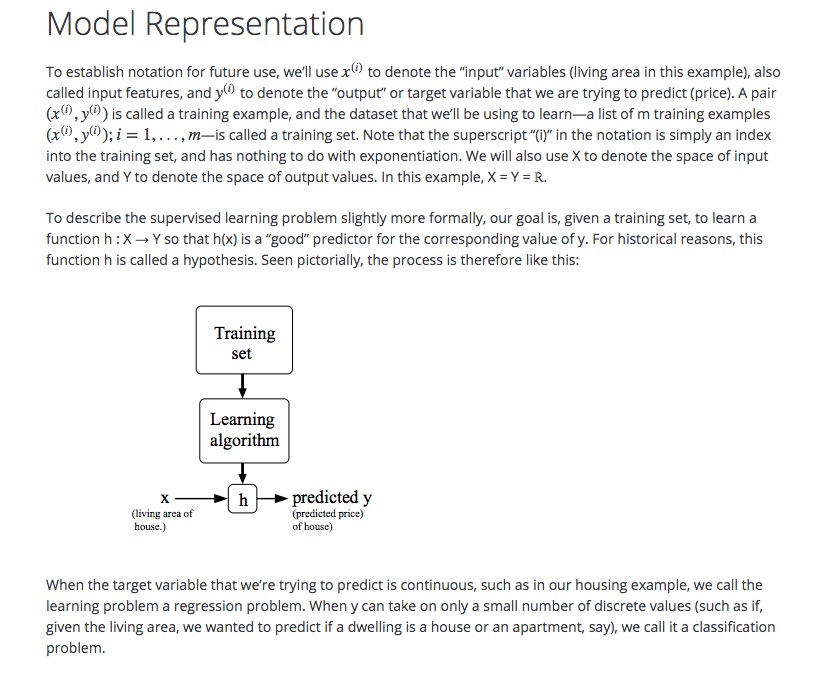
Algorithms:

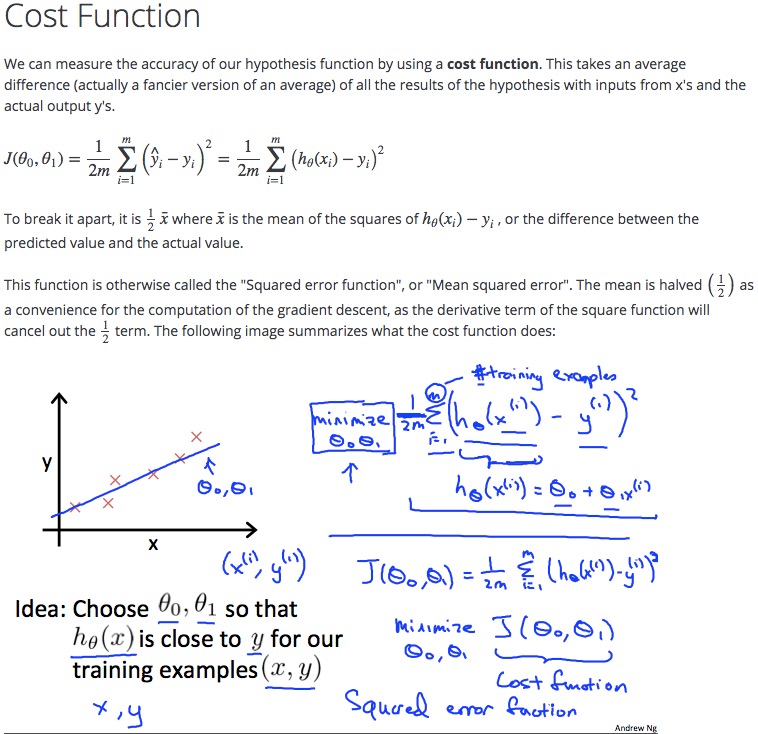
1. Linear regression
2. Logistic regression
3. Decision tree
4. SVM
5. Naïve Bayes
6. KNN
7. K-means
8. Random Forest
9. Dimensionality Reduction Algorithms
10. Gradient Boost & Adaboost

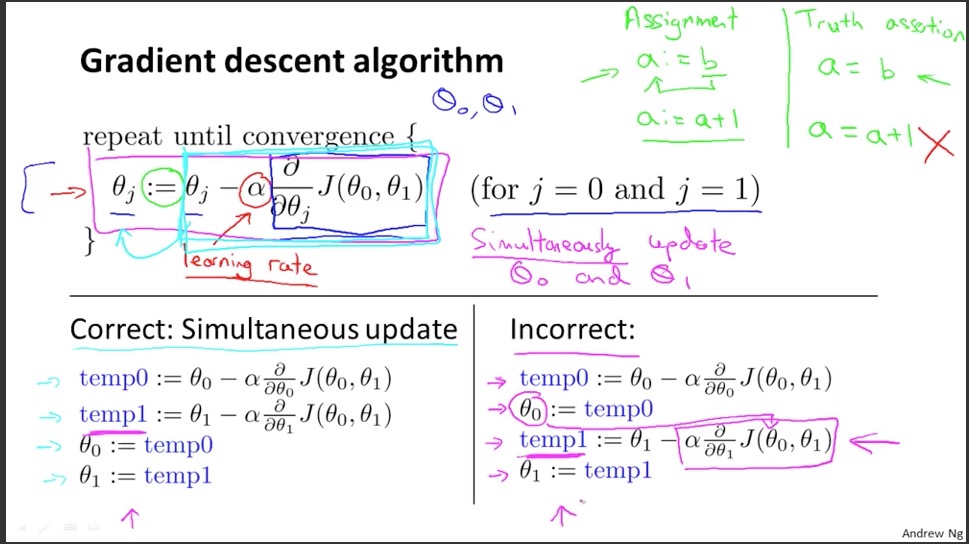
Regression: value can be numeric, Boolean etc. e.g housing px vs size, tumor nature vs size

ML: algorithm to create clusters

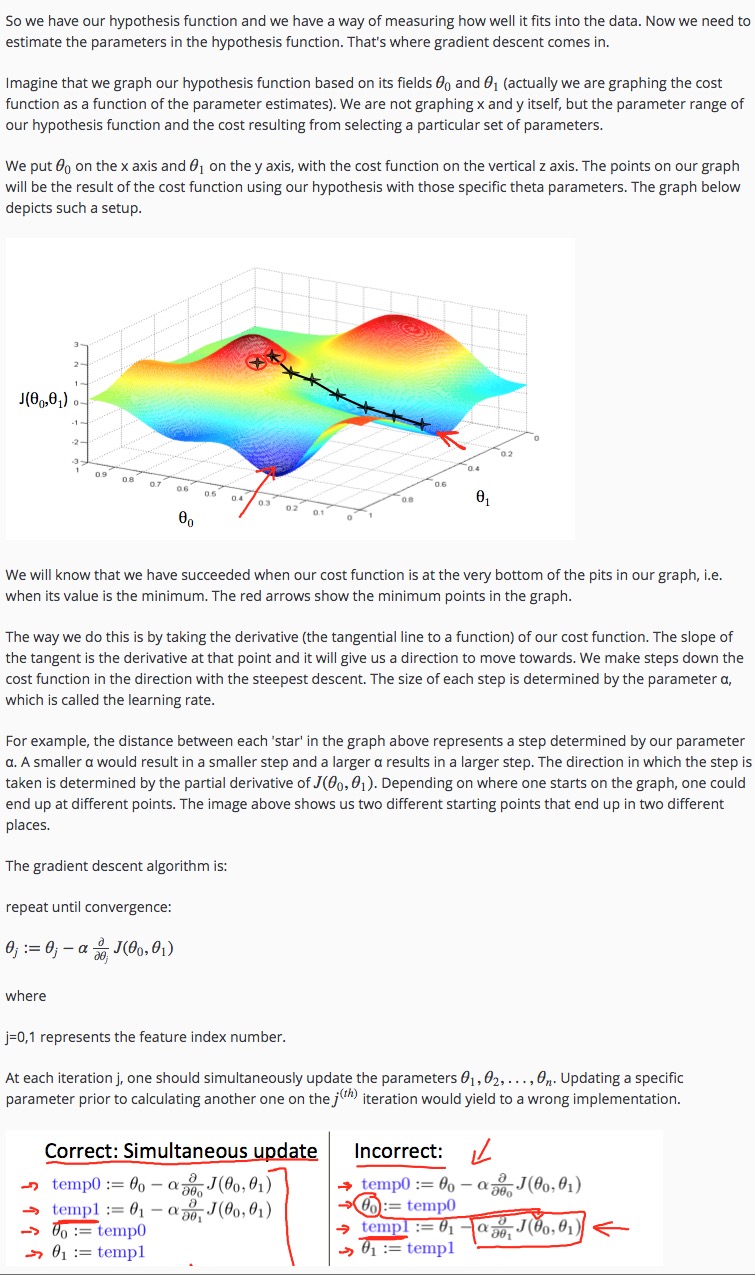
Generally, 2 problems: classification and regression

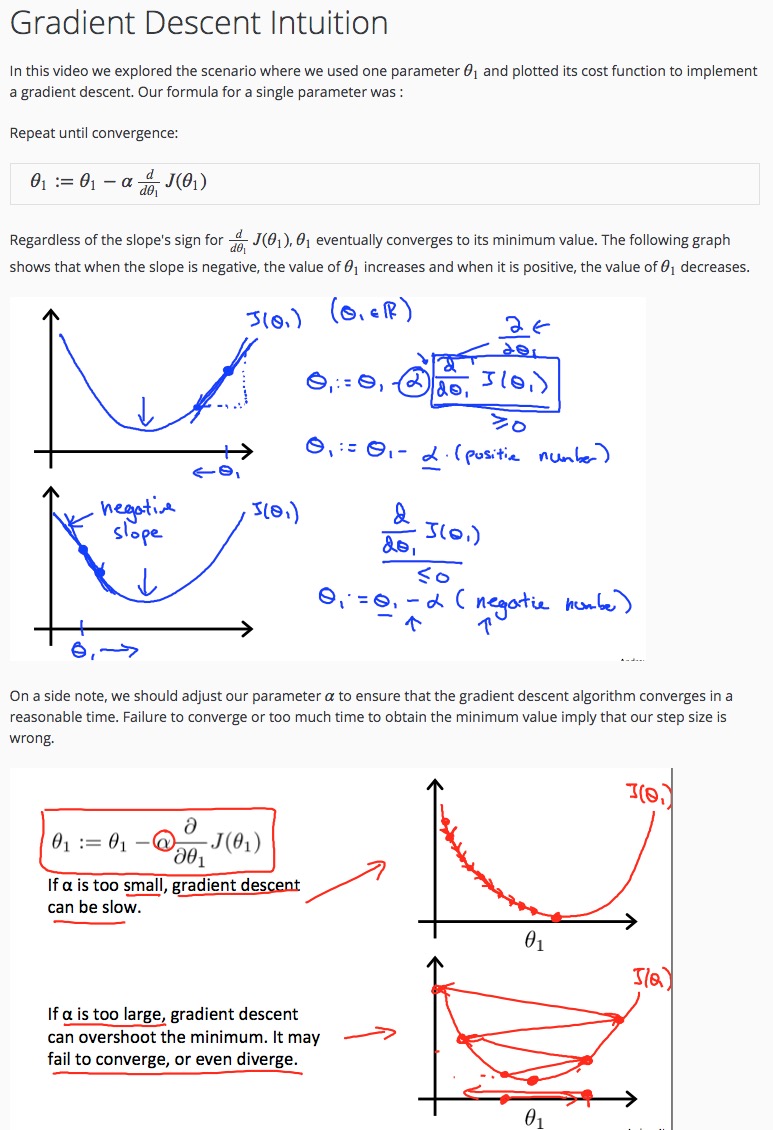


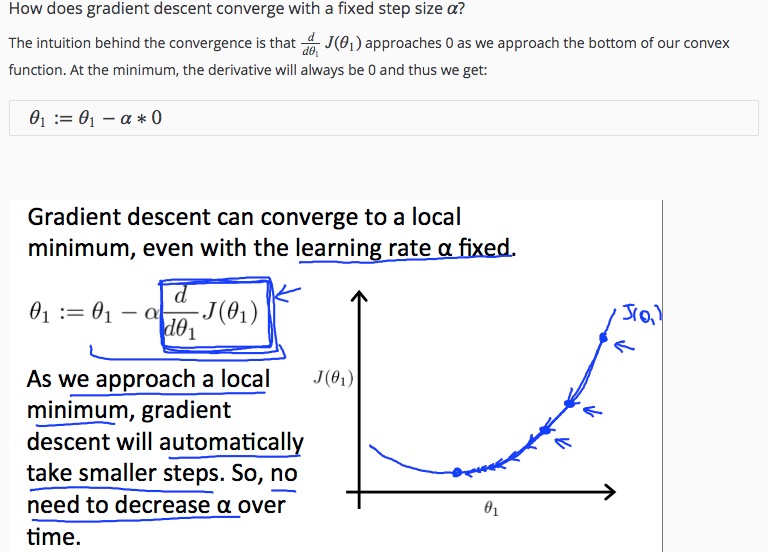


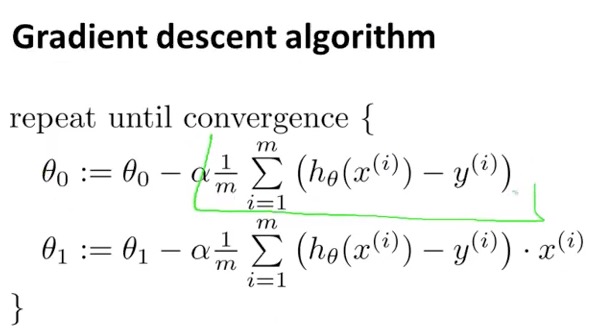


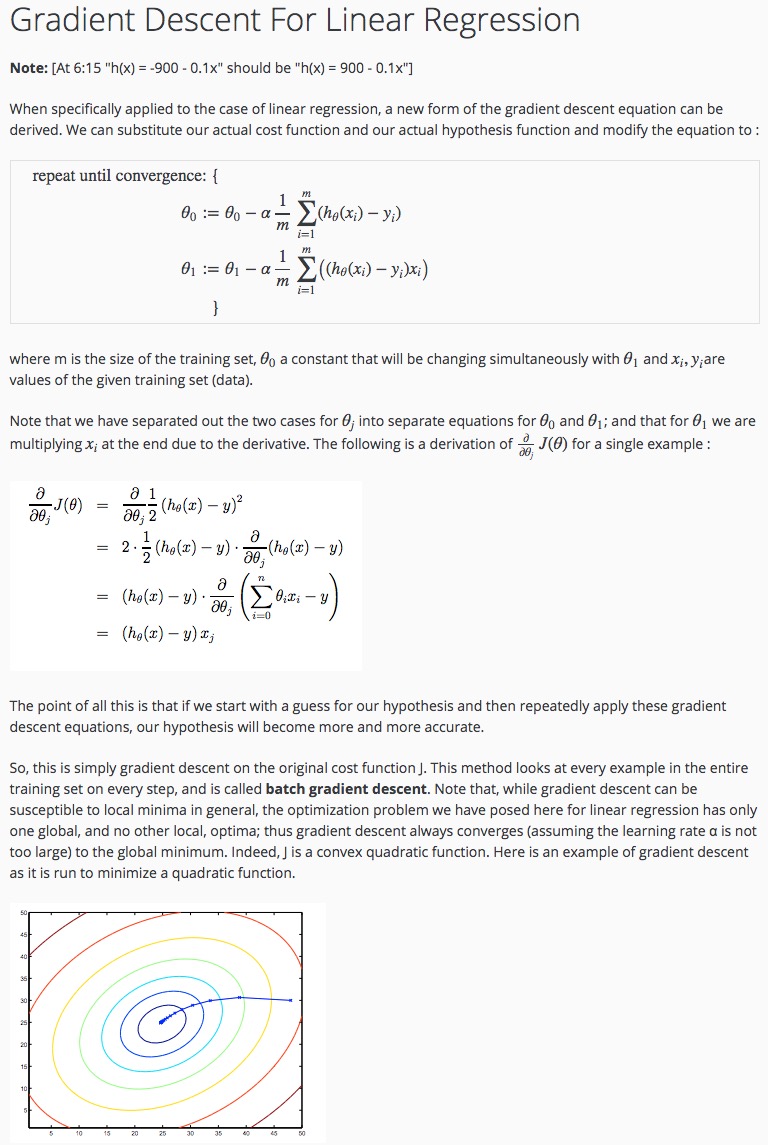
Gradient Descent:

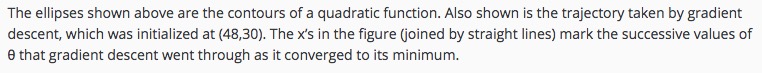


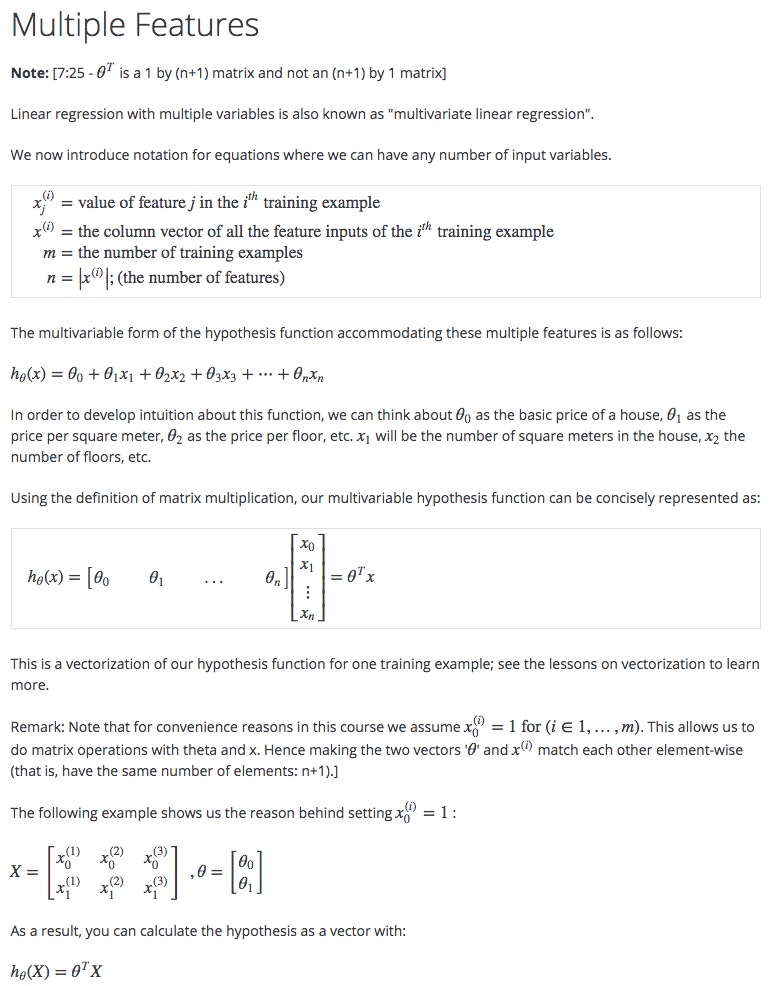


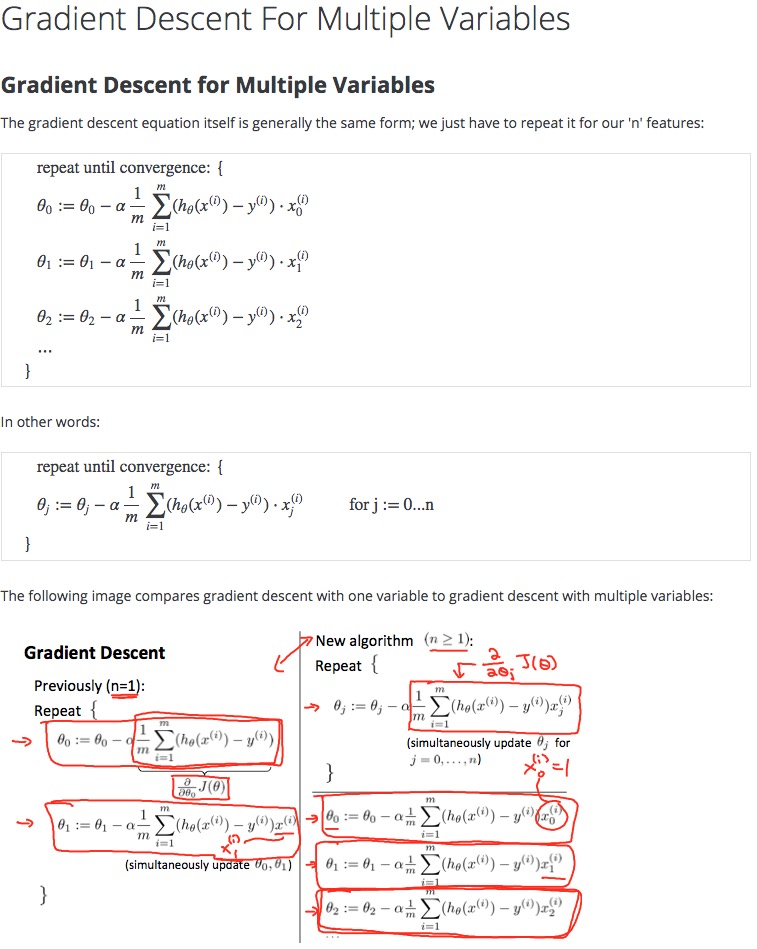


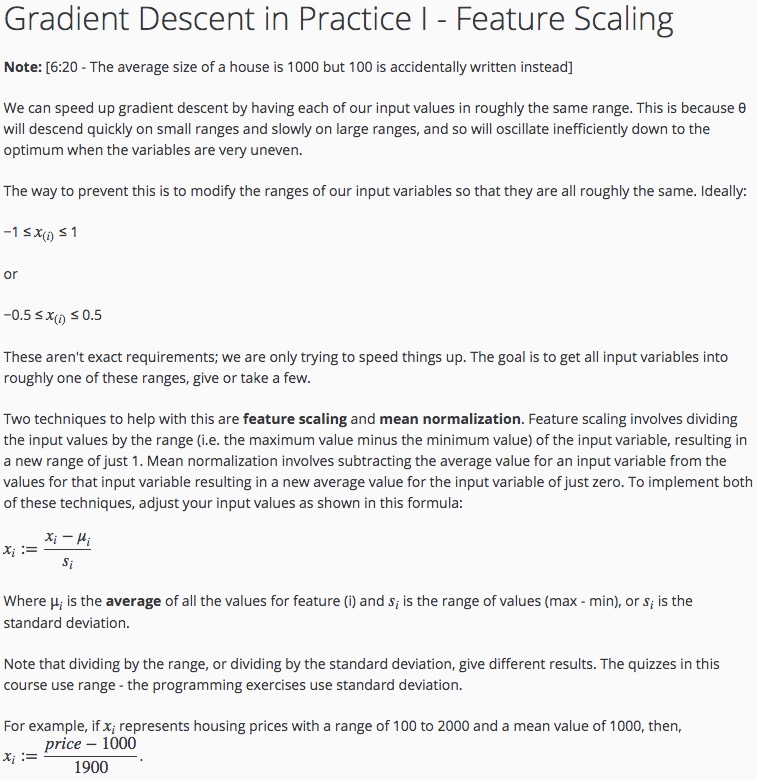


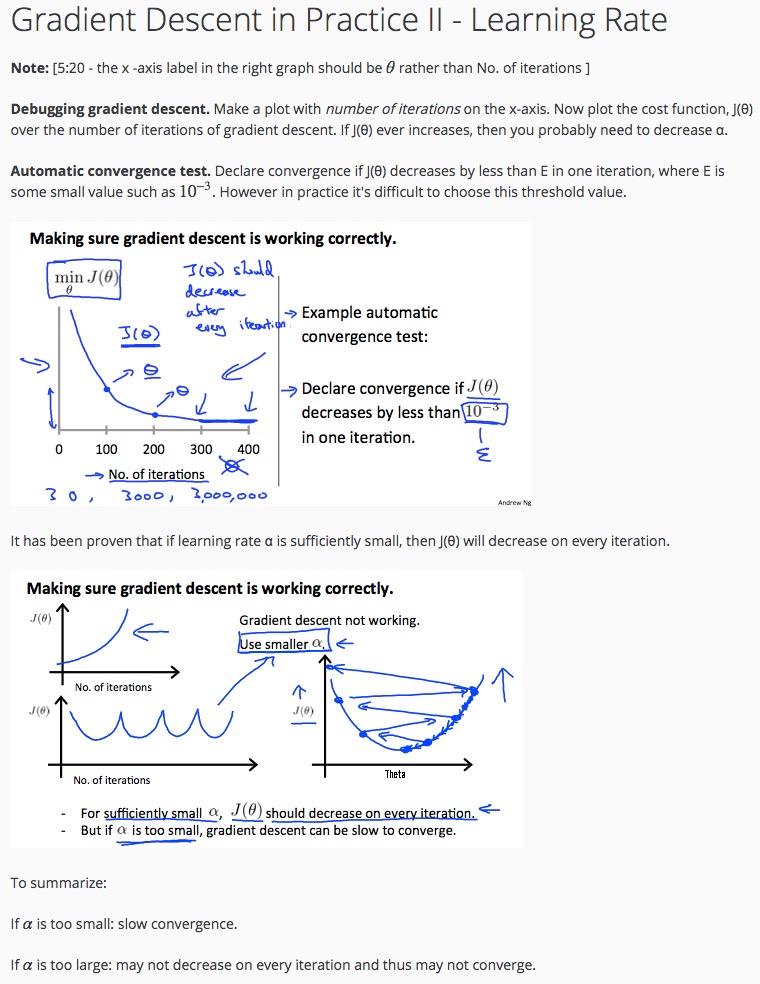


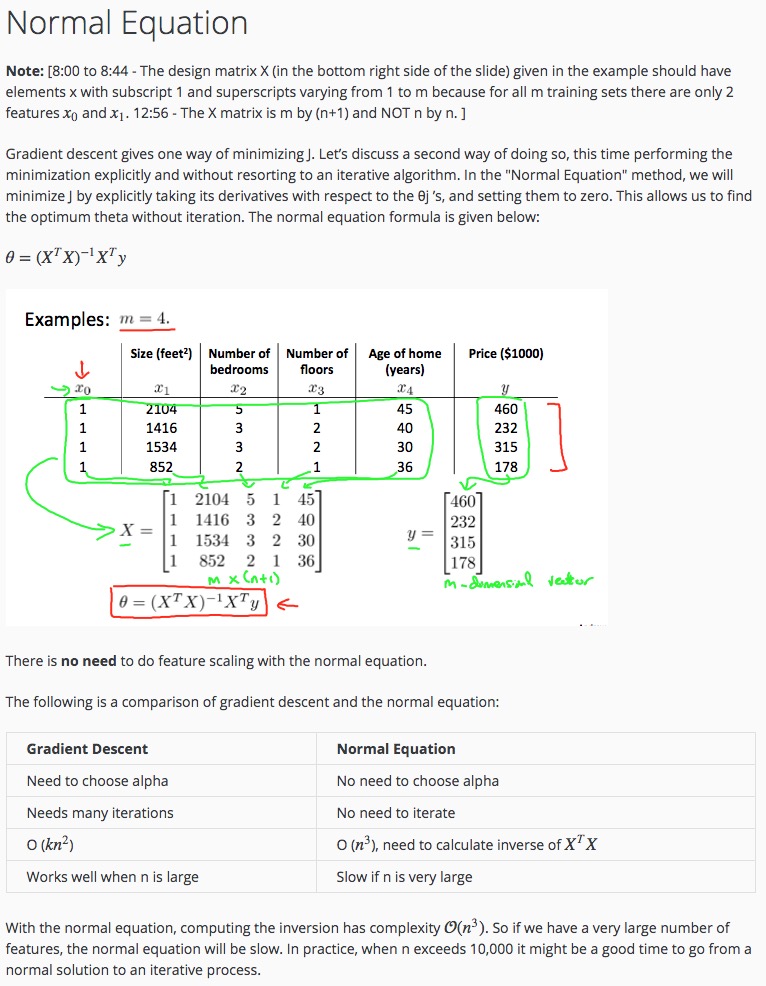


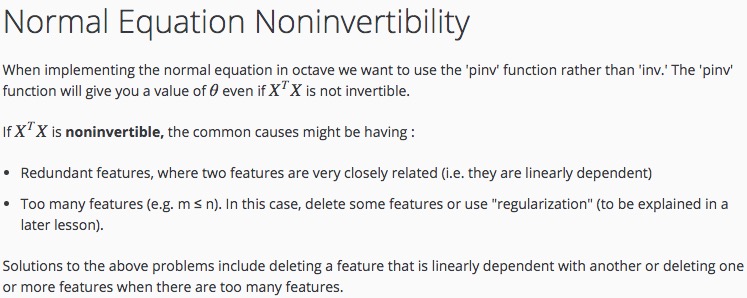


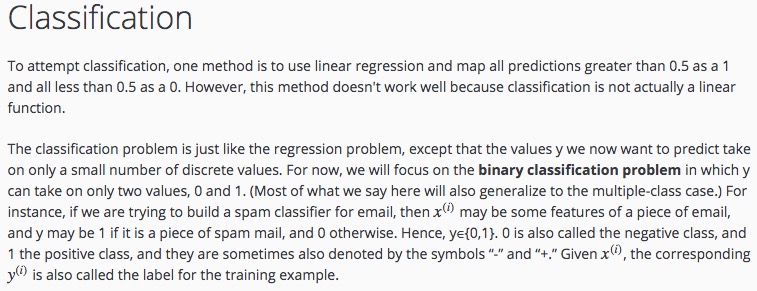


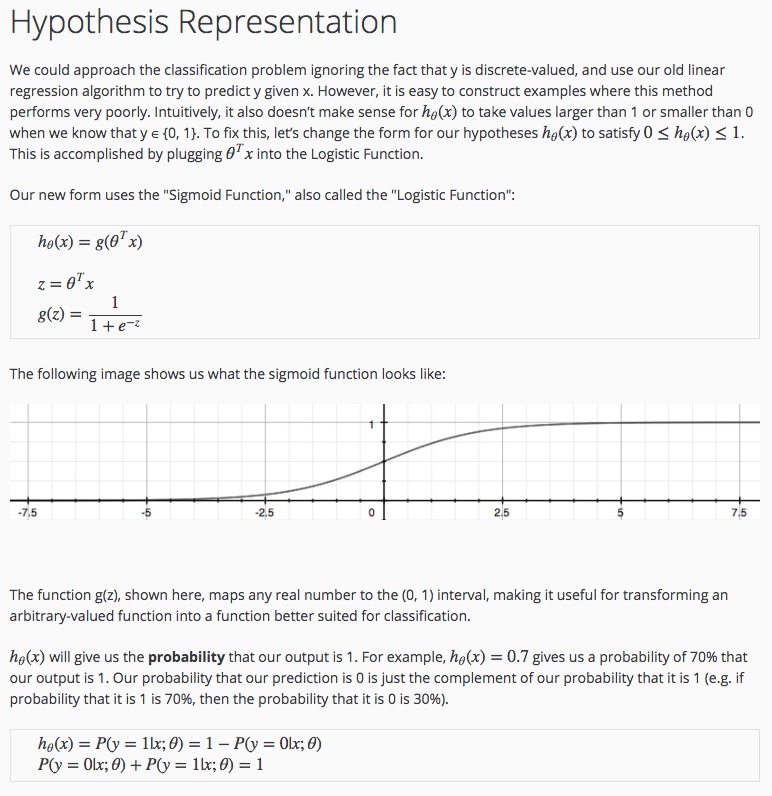


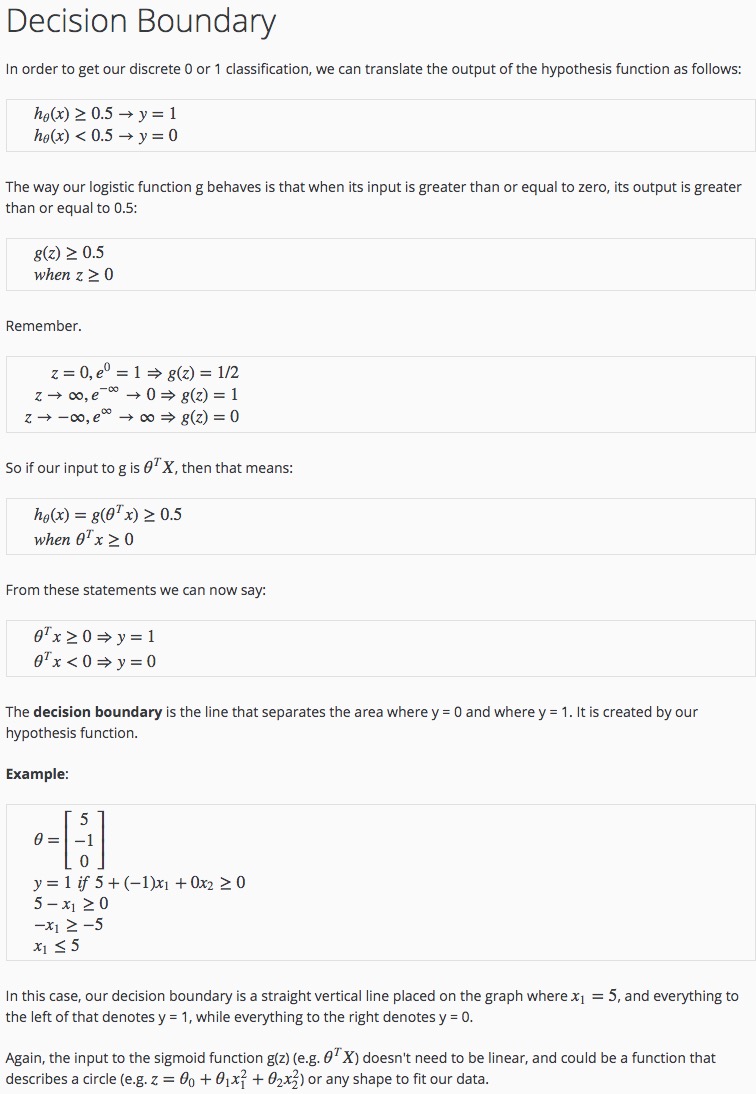


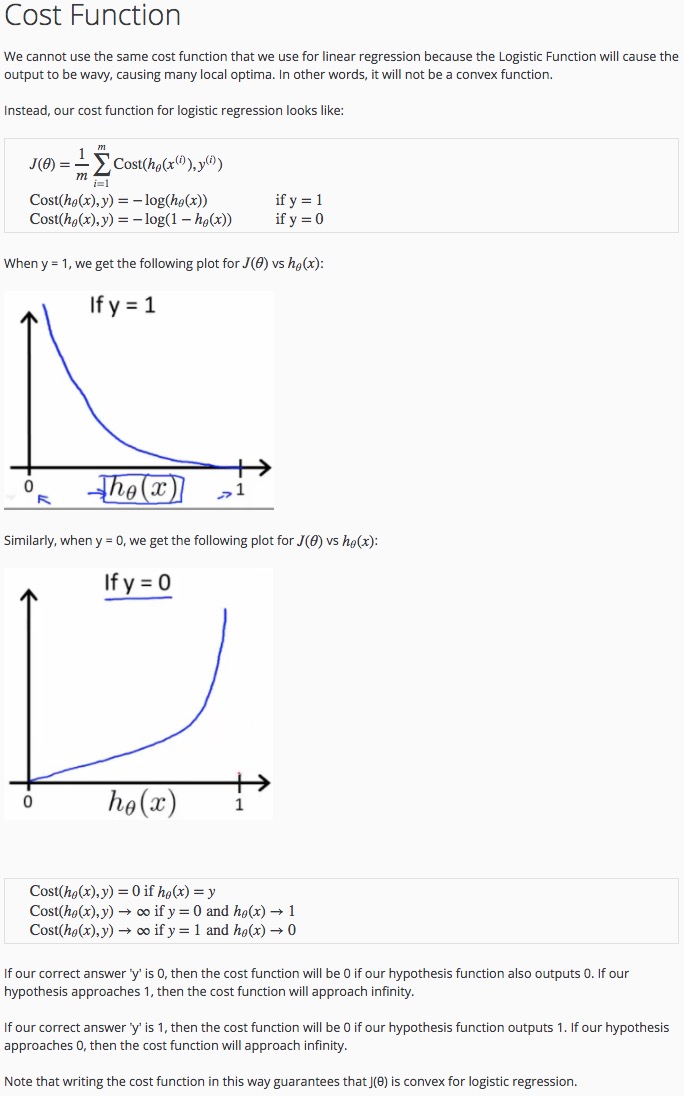


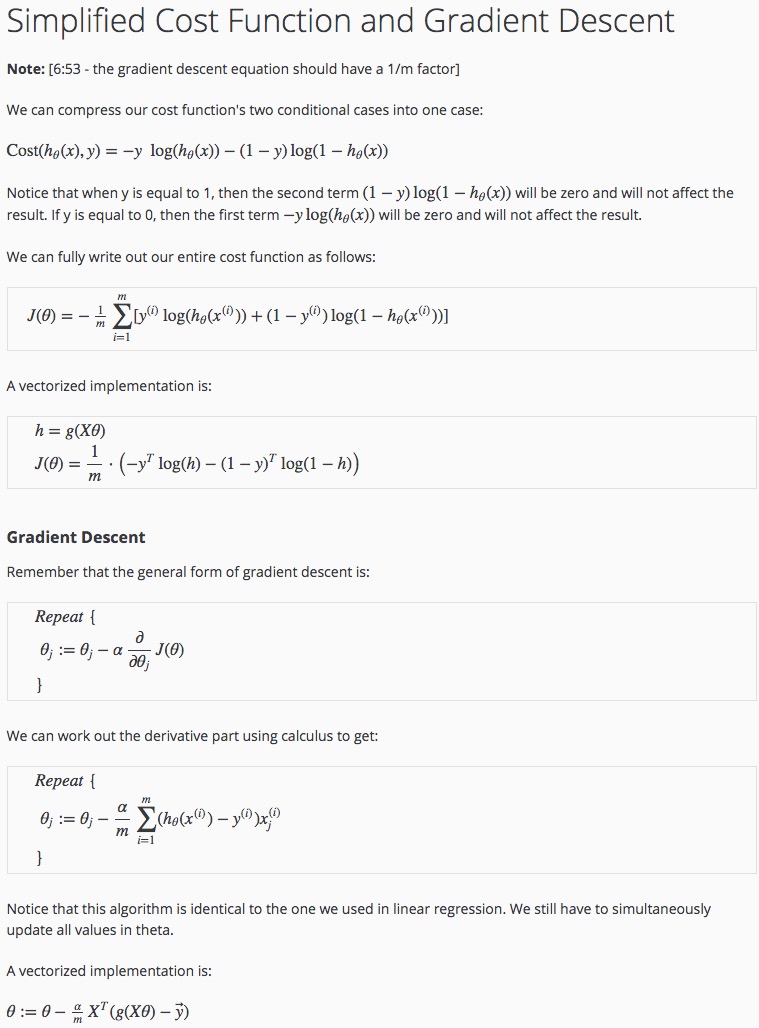


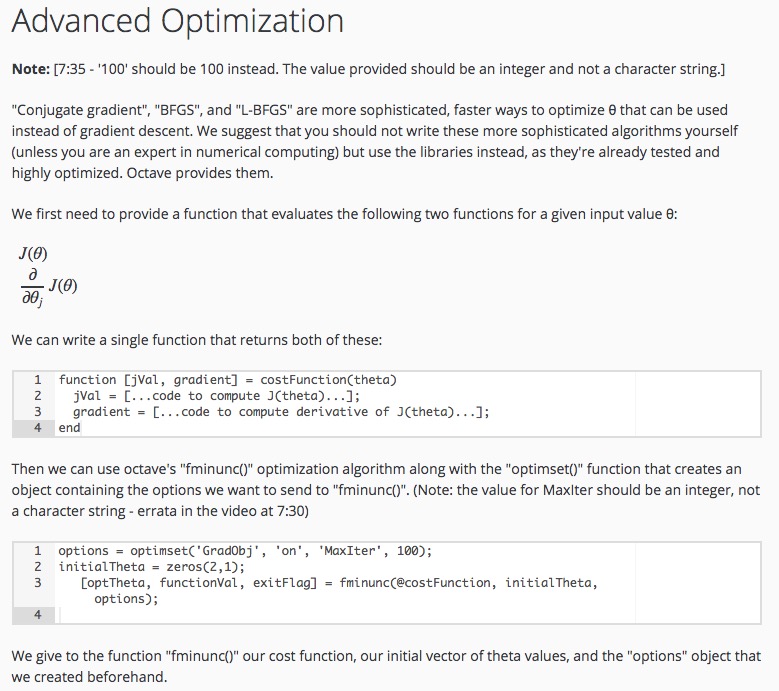












# Week 5

Unrolling parameters:

🡪 Theta1 = ones(10, 11)

Theta2 = 2 \* ones(10, 11)

Theta3 = 3 \* ones(1, 11)

thetaVec = [Theta1(:);Theta(:);Theta3(:)];

reshape(thetaVec(1:110)) 🡪 gives back theta1

