

# Introduction to Machine Learning and Data Mining Lecture-1: Introduction and Python Tutorial

Prof. Eugene Chang

# Agenda

- Class logistics
- Introduction to Machine Learning
- Anaconda Installation
- Python Tutorial

# About Myself

- Me: Yuh-Lin Eugene Chang, originally from Taiwan
- MS from UCSB, PhD Computer Engineering from UT Austin
- >20 years industry R&D (1993–now)
  - Big companies such as Panasonic and Intel
  - Small startups in SOC
  - Currently CTO of emReal Corp., working on e-commerce and mobile software development
- Background
  - Imaging systems: webcam, smartphone camera
  - Computer Vision
  - Mobile software development

# Class Structure and Policies

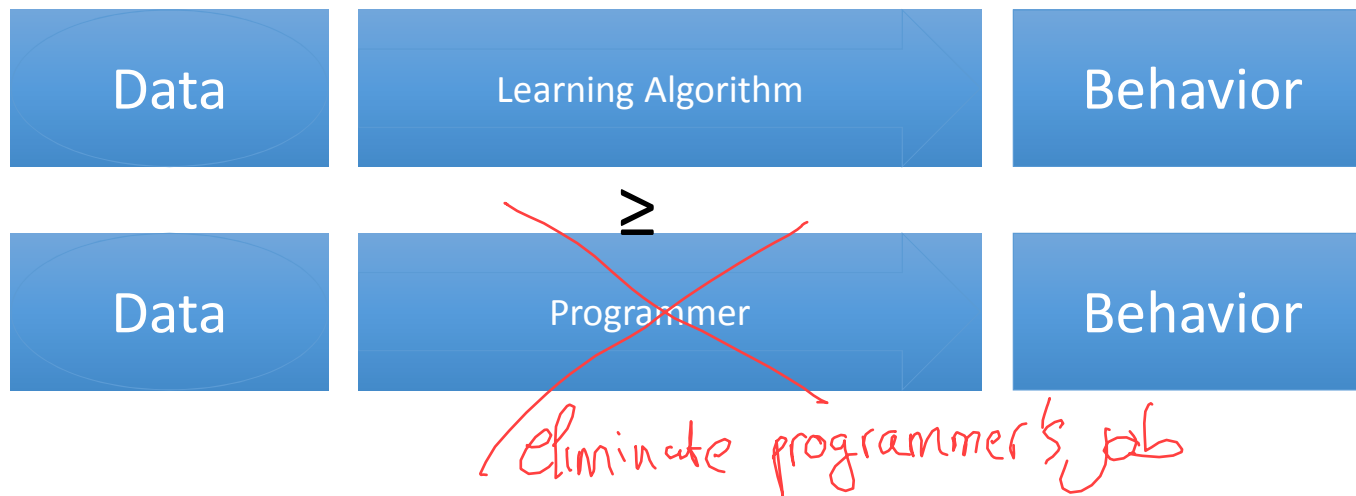
- Textbooks
  - **Machine Learning: An Algorithmic Perspective, Second Edition** by Stephen Marsland. **Publisher:** Chapman and Hall/CRC (October 8, 2014). **ISBN-10:** 1466583282. **ISBN-13:** 978-1466583283
- Course focus
  - Machine Learning basic theories (lots of Math)
  - Python tools and programming (lots of coding)
  - Team project instead of final exam (specify, design, implement, test, present)
- Grades
  - Homework 30%, Midterm 30%, final project 40%
- Course website
  - SVU FTP: [http://class.svuca.edu/~eugene.chang/class/CS596-029\\_2015\\_Summer/](http://class.svuca.edu/~eugene.chang/class/CS596-029_2015_Summer/)
  - Download Anaconda now: <http://continuum.io/downloads> or the above site

# Teaching Plan

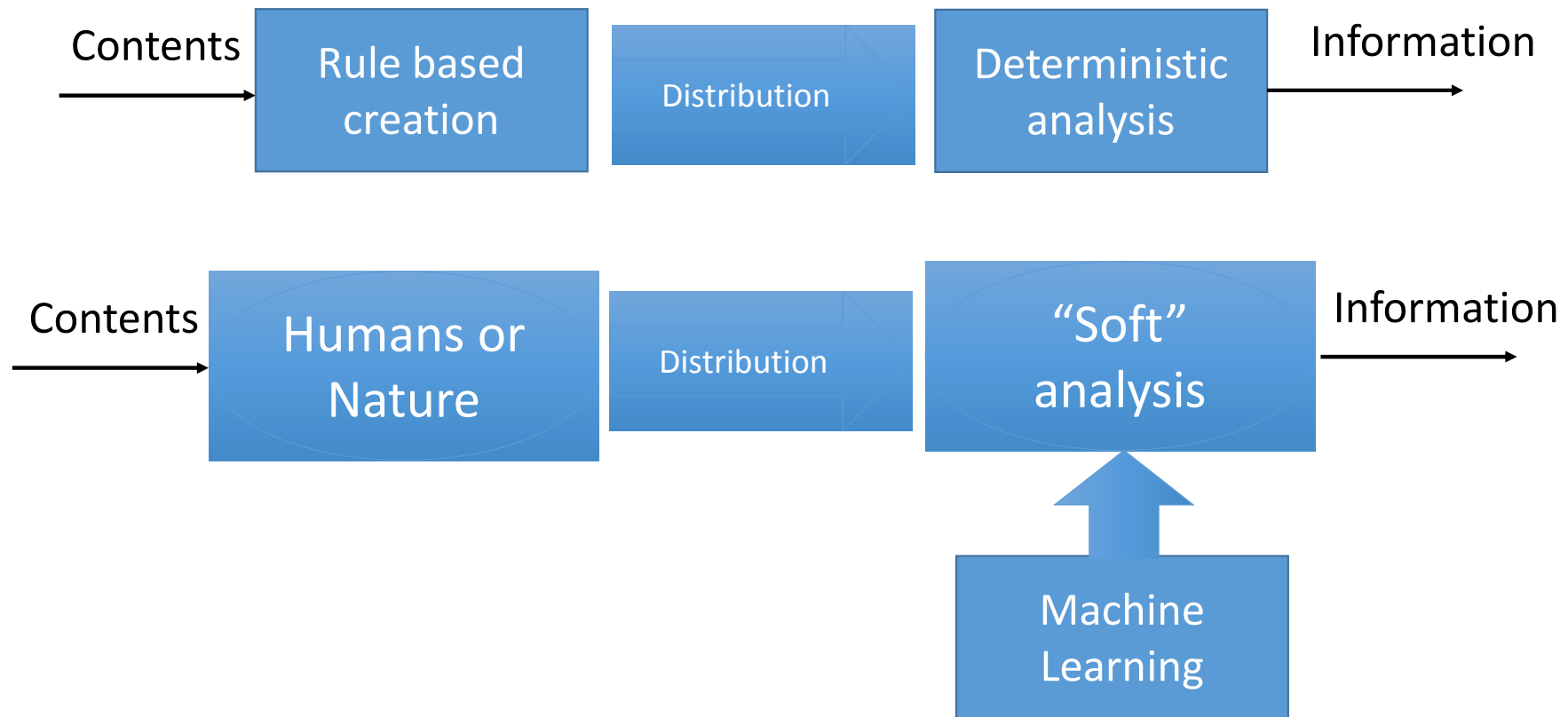
Week	Topic	Reading/Homework/Case Assignment
• 1 (May 15)	Introduction, Python Tutorial	Ch. 1, Ch. 2
• 2	Math Primer, Linear Regression	Ch. 3, Class Notes
• 3	Decision Trees, SK-learn	Ch. 12 (HW-1)
• 4	Neural Networks	Ch. 3, Ch. 4, Ch. 5
• 5	Support Vector Machine, Kernel Methods	Ch. 8 (HW-2)
• 6	Probabilistic Learning and Ensemble Methods	Ch. 7, Ch. 13
• 7	Unsupervised Learning, Project Topics	Ch. 14 (HW-3)
• 8	Dimensionality Reduction, Midterm Review	Ch. 6
• 9 (July 10)	<b><u>Mid-Term Examination</u></b>	
• 10	Data Mining, Project Selection	Class Notes
• 11	Data Mining, Project Proposal	Class Notes, (HW-4)
• 12	Semi-Supervised Learning, Project Progress	Class Notes
• 13	Object Recognition, Project Progress	Class Notes
• 14	Bioinformatics, Project Progress	Class Notes
• 15 (Aug 21)	<b><u>Final Project Presentation</u></b>	

# Machine Learning

- Analyze data to support decision making
- Automatically identifying patterns in data
- Automatically making decisions based on data
- Hypothesis:



# Why is Machine Learning Needed?



# Applications of Machine Learning

- Computer Vision: self-driving vehicle, security, face detection, object recognition
- Fingerprint ID
- Speech Recognition: Siri
- Machine translation: Google Translate
- Recommendation system: video, music, books, smart ads
- Internet search engine
- Fraud detection
- Weather prediction
- Stock market prediction
- Many many more



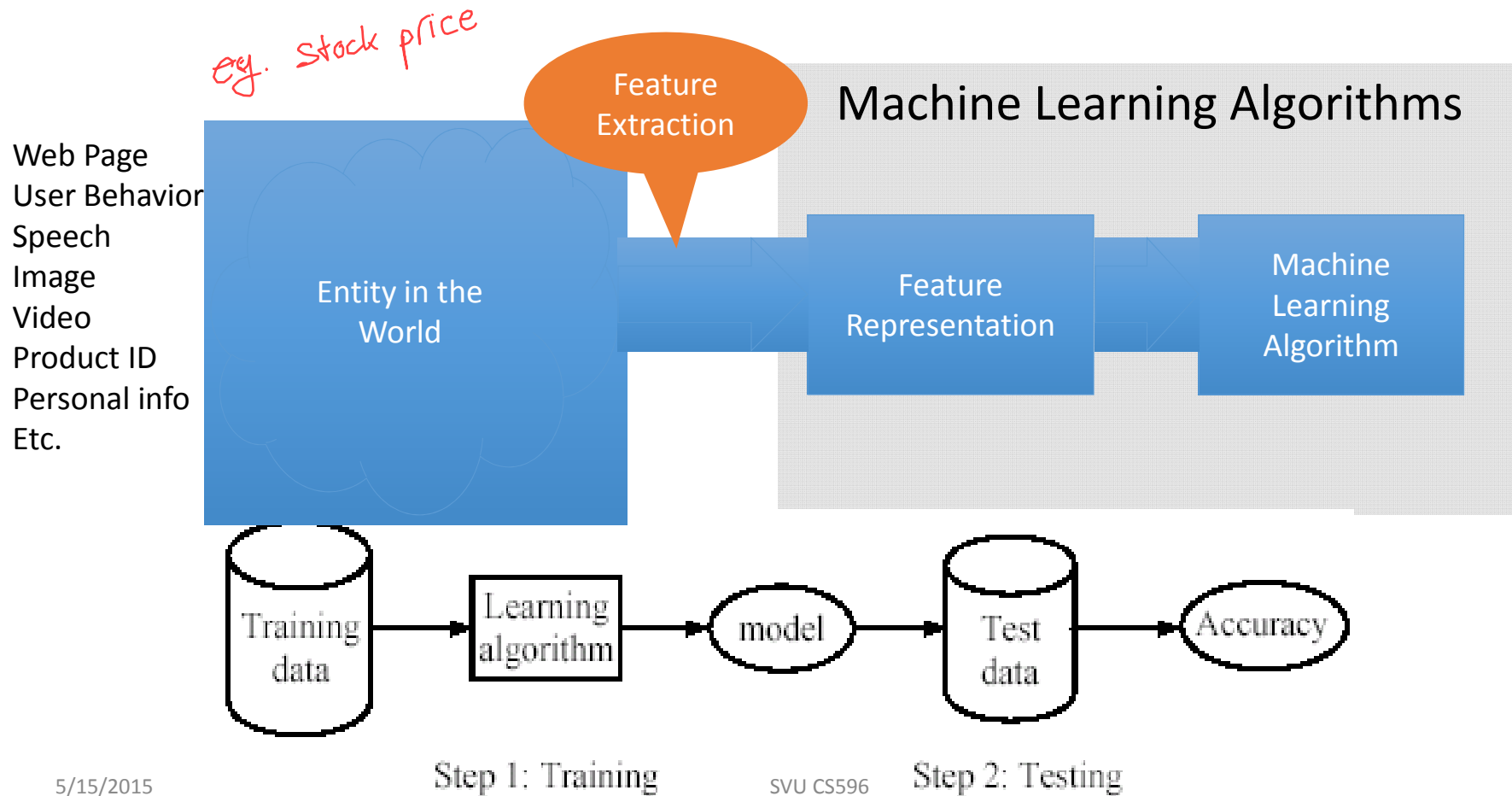
# Examples and Resources on the Web

- Google
  - <https://cloud.google.com/prediction/docs>
- Facebook
  - <https://research.facebook.com/datascience>
- Amazon Machine Learning API: service for developers to use ML
  - <http://aws.amazon.com/machine-learning/>
  - <http://aws.amazon.com/public-data-sets/>
- Udacity
  - <https://www.udacity.com/course/intro-to-machine-learning--ud120>
- Coursera
  - <https://www.coursera.org/course/machlearning>
- Kaggle.com → *host competition in ML*
  - a platform for [predictive modelling](#) and [analytics](#) competitions

} *courses.*

*(otto Group Product Classification)*

# Typical System Flow



# Major Tasks

- Data gathering
- Feature extraction
- Supervised Learning → have input data & expected result
  - Density estimate
  - Regression
    - Predict a numerical value from “other information”
  - Classification
    - Predict a categorical value
- Unsupervised Learning: Clustering → unknown result
  - Identify groups of similar entities
- Evaluation
- Semi-Supervised Learning: Reinforcement → only know if the outcome is correct or incorrect w/o specific details

# Example: Feature Representations

Height	Weight	Eye Color	Gender
66	170	Blue	Male
73	210	Brown	Male
72	165	Green	Male
70	180	Blue	Male
74	185	Brown	Male
68	155	Green	Male
65	150	Blue	Female
64	120	Brown	Female
63	125	Green	Female
67	140	Blue	Female
68	165	Brown	Female
66	130	Green	Female

# Classification

- Identify which of  $N$  classes a data point,  $\mathbf{x}$ , belongs to.
- $\mathbf{x}$  is a column vector of features.

$$\vec{x} = \begin{pmatrix} x_0 \\ x_1 \\ \dots \\ x_{n-1} \end{pmatrix} \quad \text{OR} \quad \vec{x} = \begin{pmatrix} f_0(x) \\ f_1(x) \\ \dots \\ f_{m-1}(x) \end{pmatrix}$$

# Target Values

- In **supervised** approaches, in addition to a data point,  $\mathbf{x}$ , we will also have access to a target value,  $\mathbf{t}$ .

## Goal of Classification

Identify a function  $y$ , such that  $y(\mathbf{x}) = \mathbf{t}$

# Feature Representations

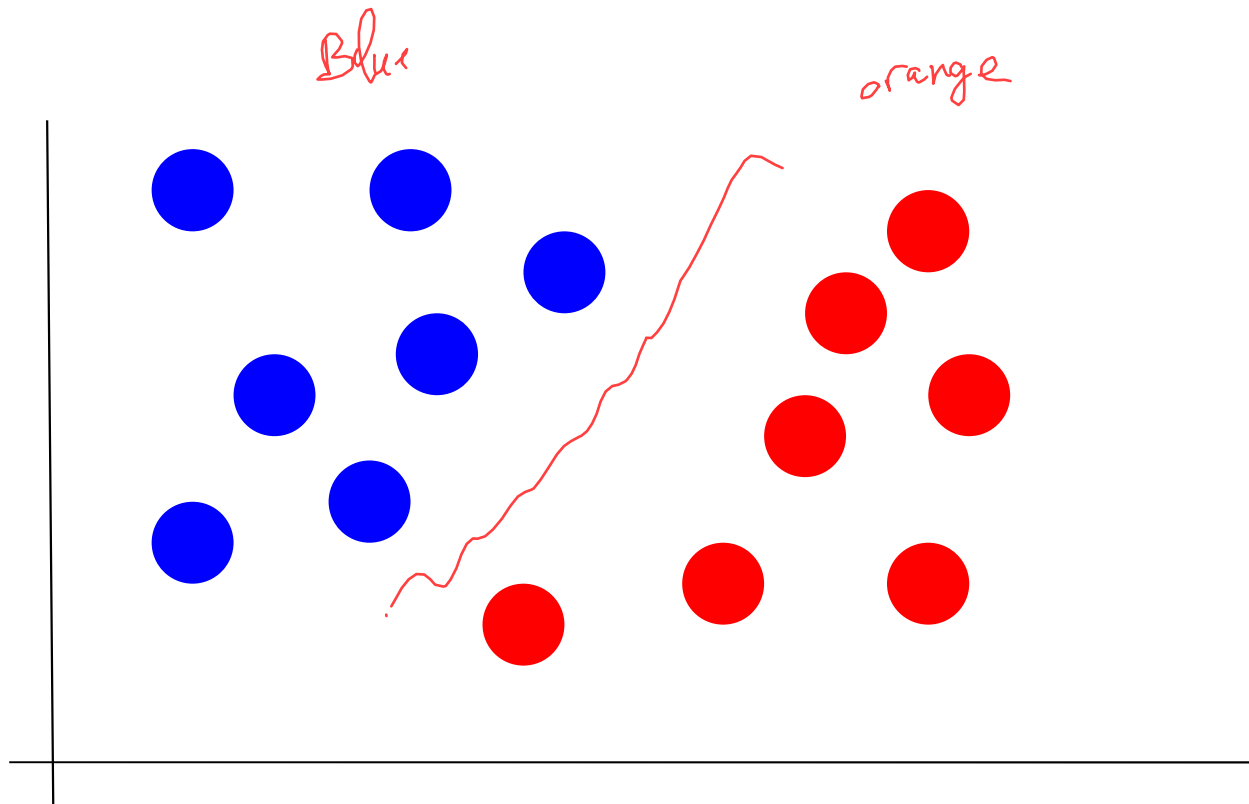
$$\vec{x}_0 = \begin{pmatrix} 66 \\ 170 \\ \text{Blue} \end{pmatrix}$$

training set data

Height	Weight	Eye Color	Gender
66	170	Blue	Male
73	210	Brown	Male
72	165	Green	Male
70	180	Blue	Male
74	185	Brown	Male
68	155	Green	Male
65	150	Blue	Female
64	120	Brown	Female
63	125	Green	Female
67	140	Blue	Female
68	165	Brown	Female
66	130	Green	Female

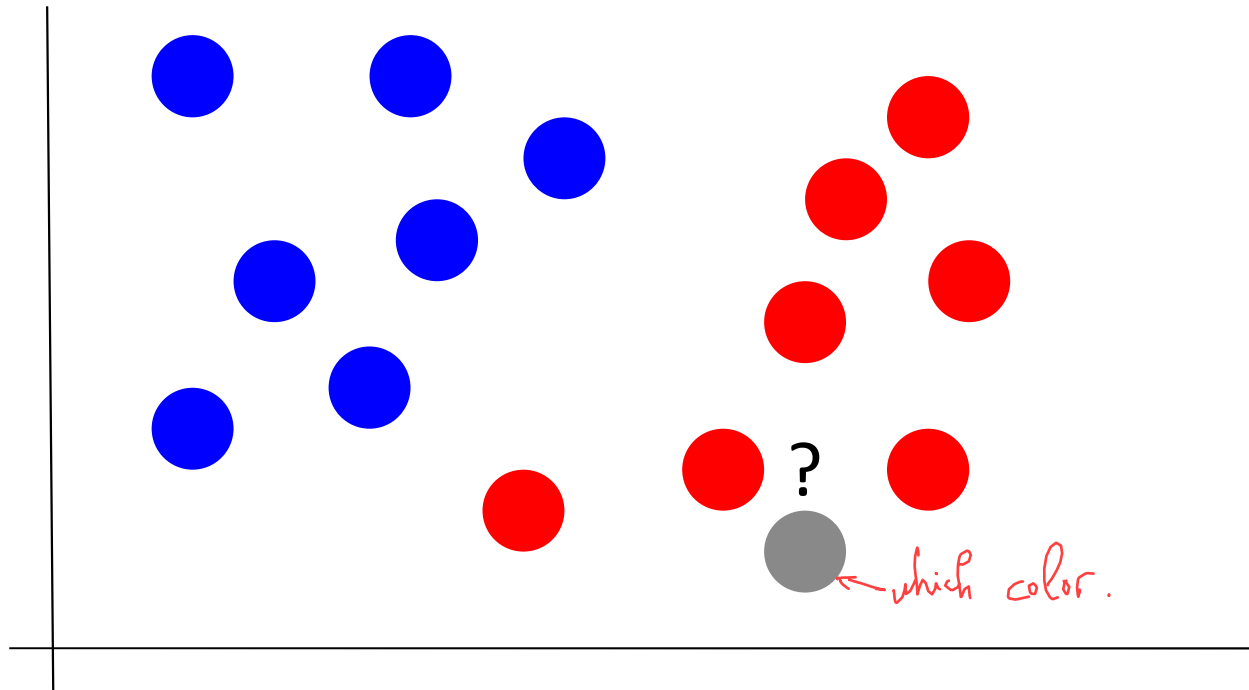
$t_0 = \text{Male}$

# Graphical Example of Classification

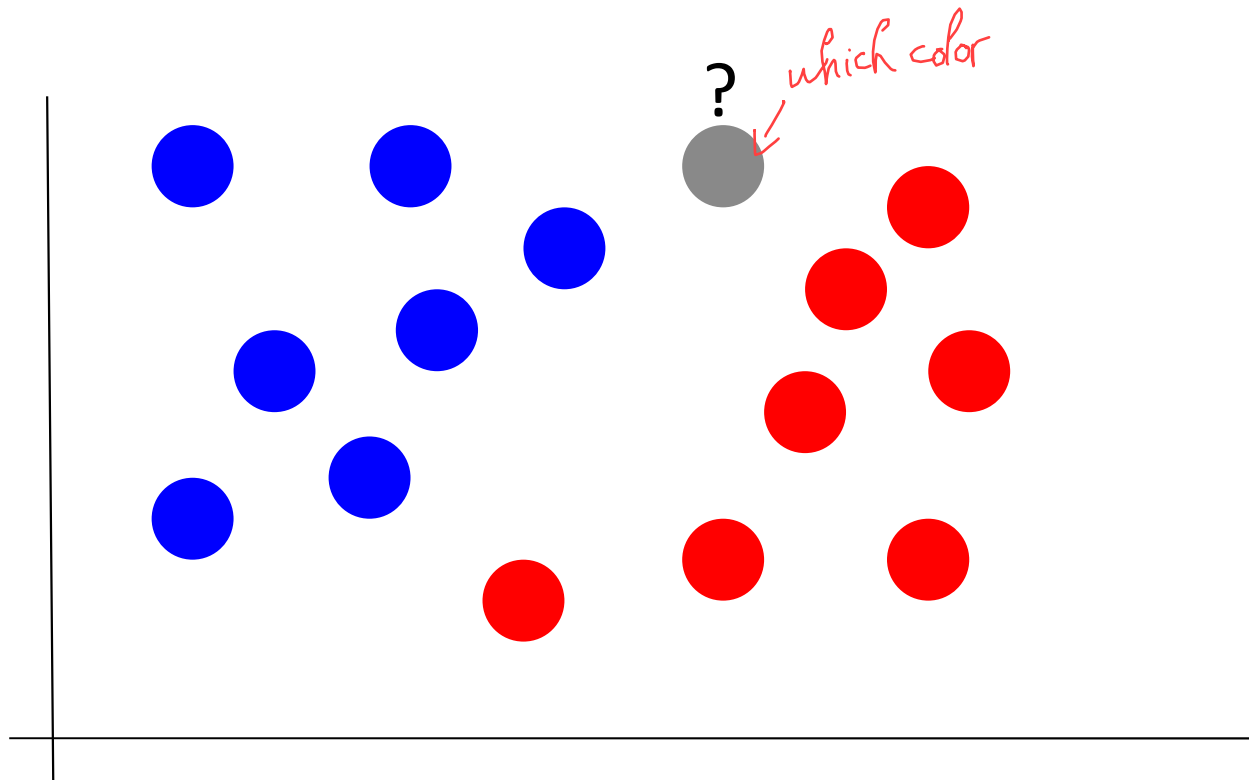




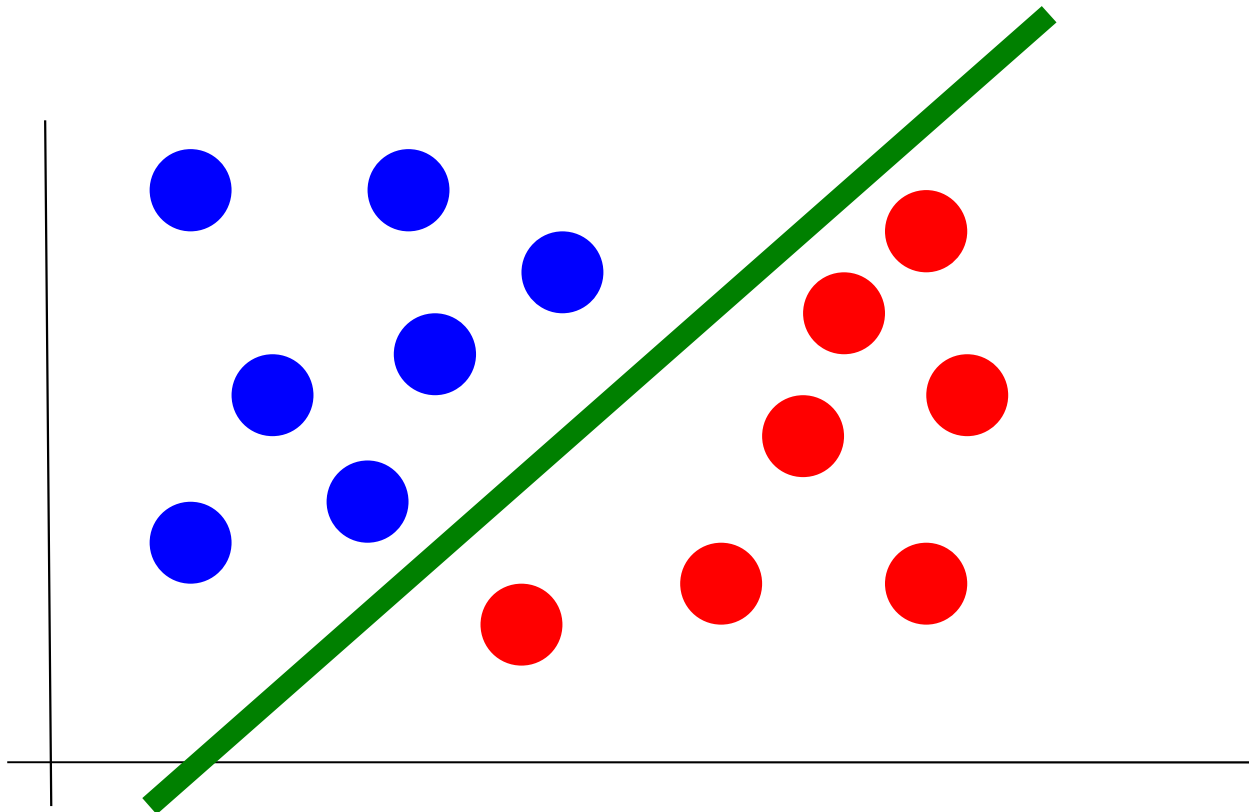
# Graphical Example of Classification



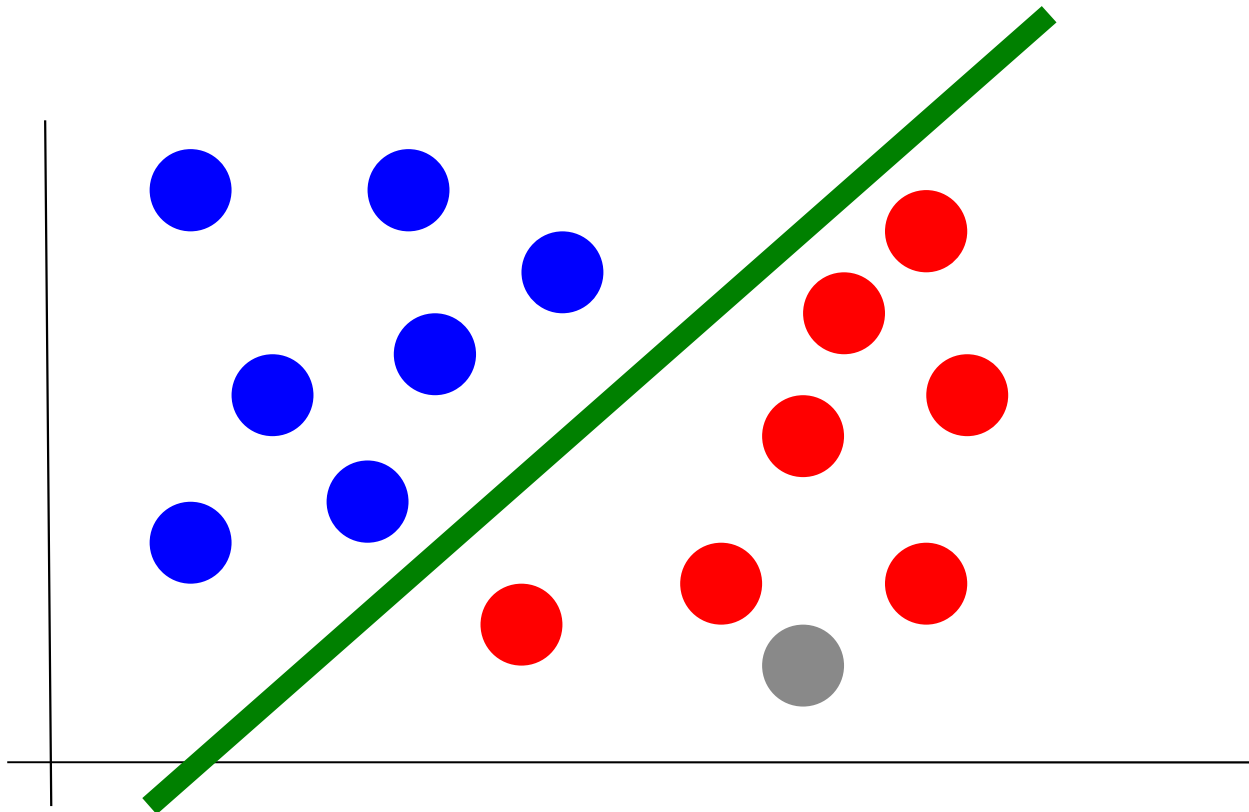
# Graphical Example of Classification



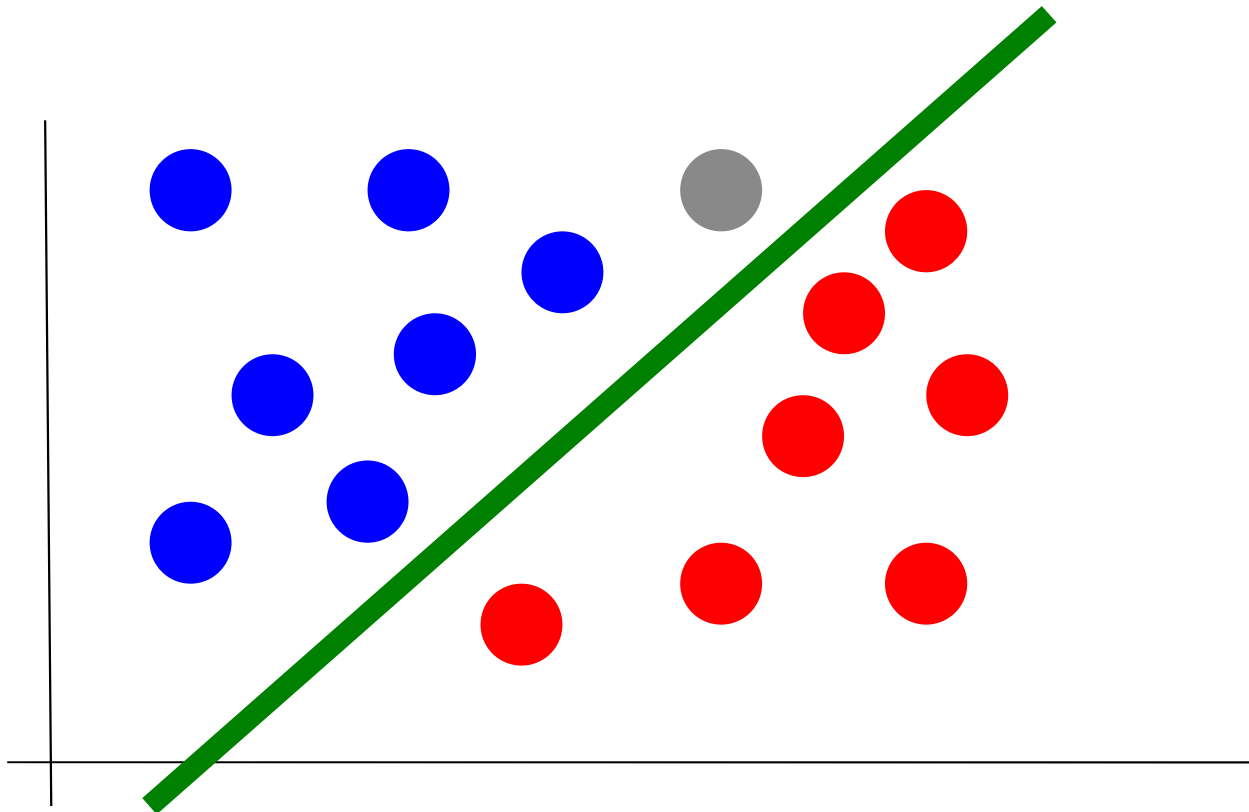
# Graphical Example of Classification



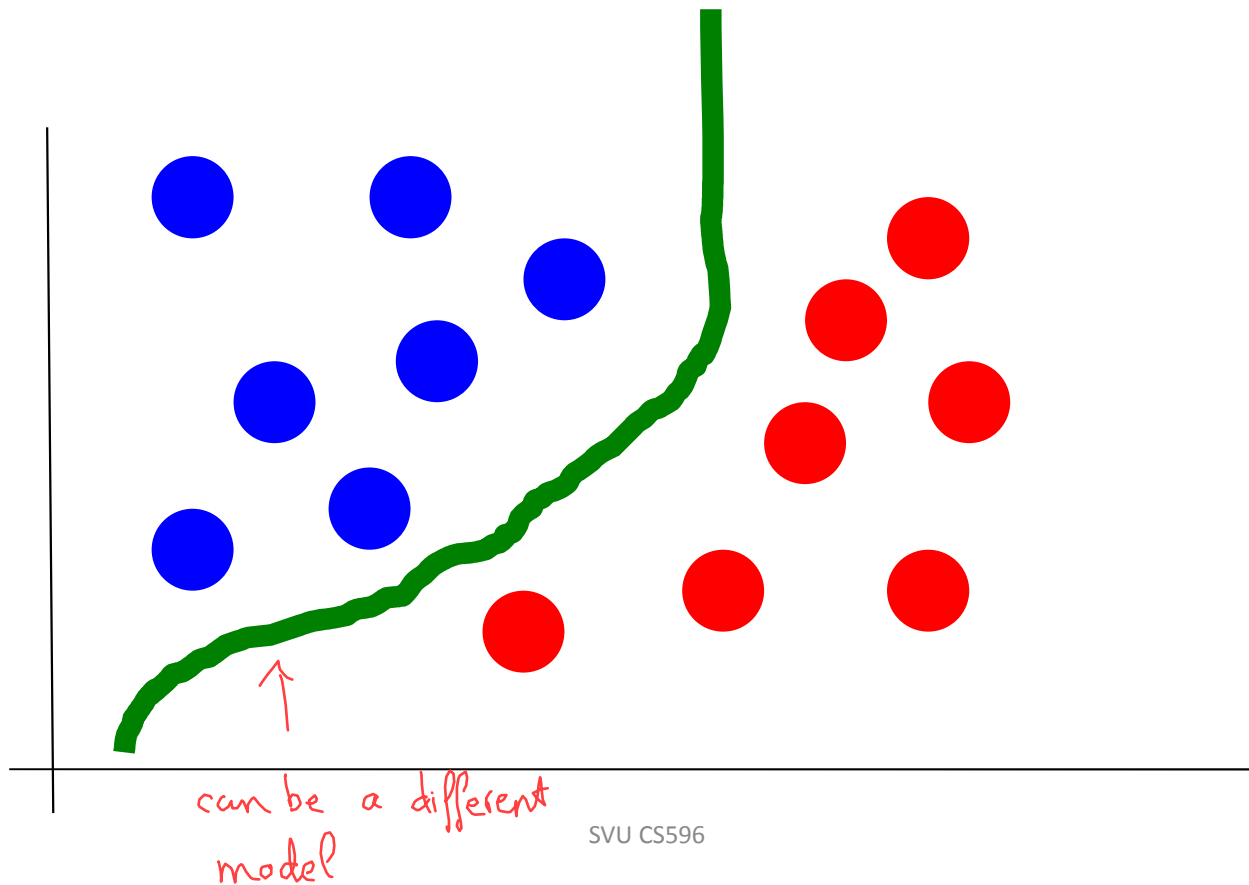
# Graphical Example of Classification



# Graphical Example of Classification



# Decision Boundaries



# Regression

- Regression is a **supervised** machine learning task.
  - So a target value, **t**, is given.
- Classification: nominal **t**  $t \in \{c_0, \dots, c_{N-1}\}$
- Regression: continuous **t**  $t \in \mathbb{R}$

## Goal of Classification

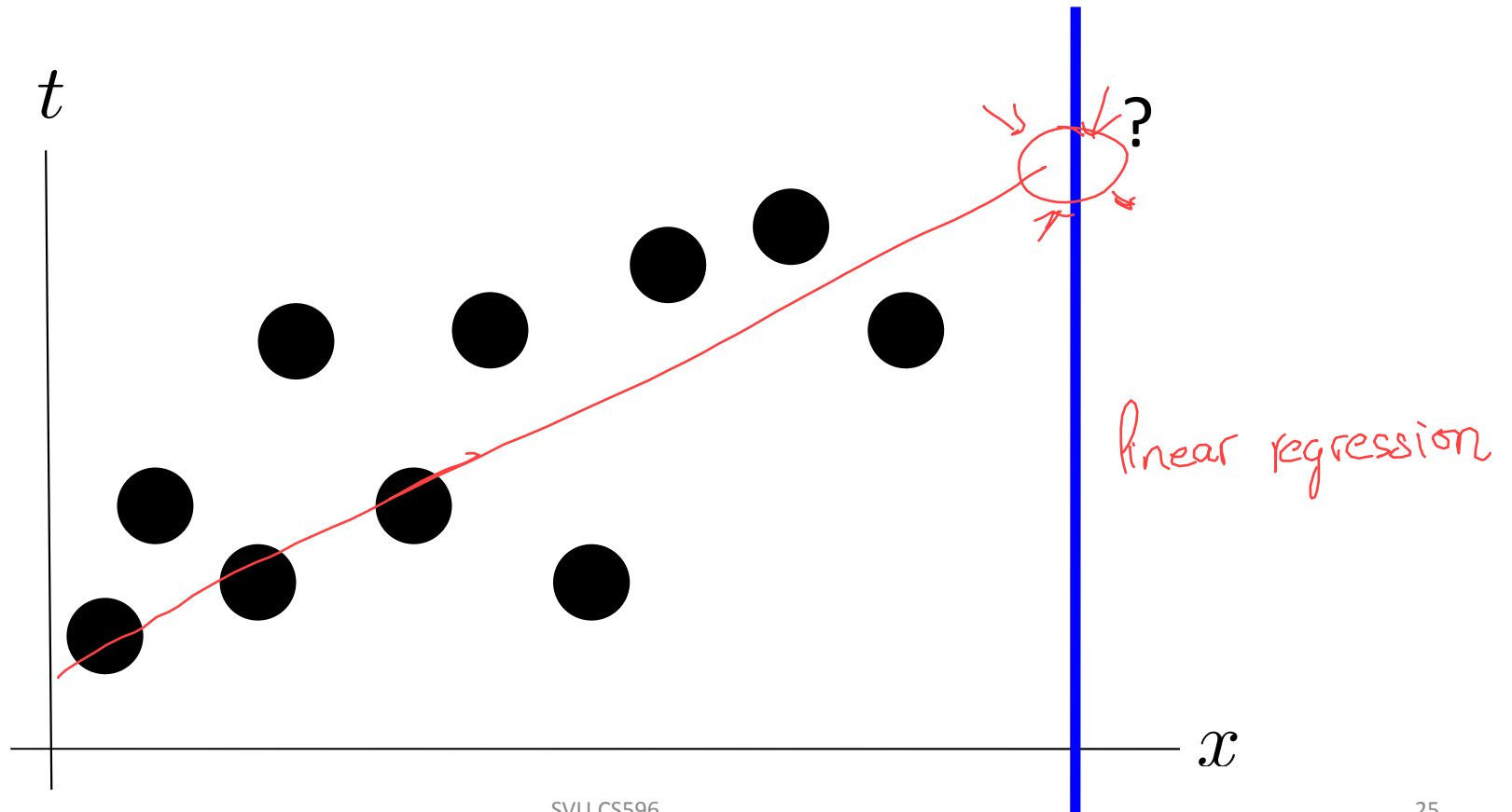
Identify a function  $y$ , such that  $y(\mathbf{x}) = \mathbf{t}$

# Differences between Classification and Regression

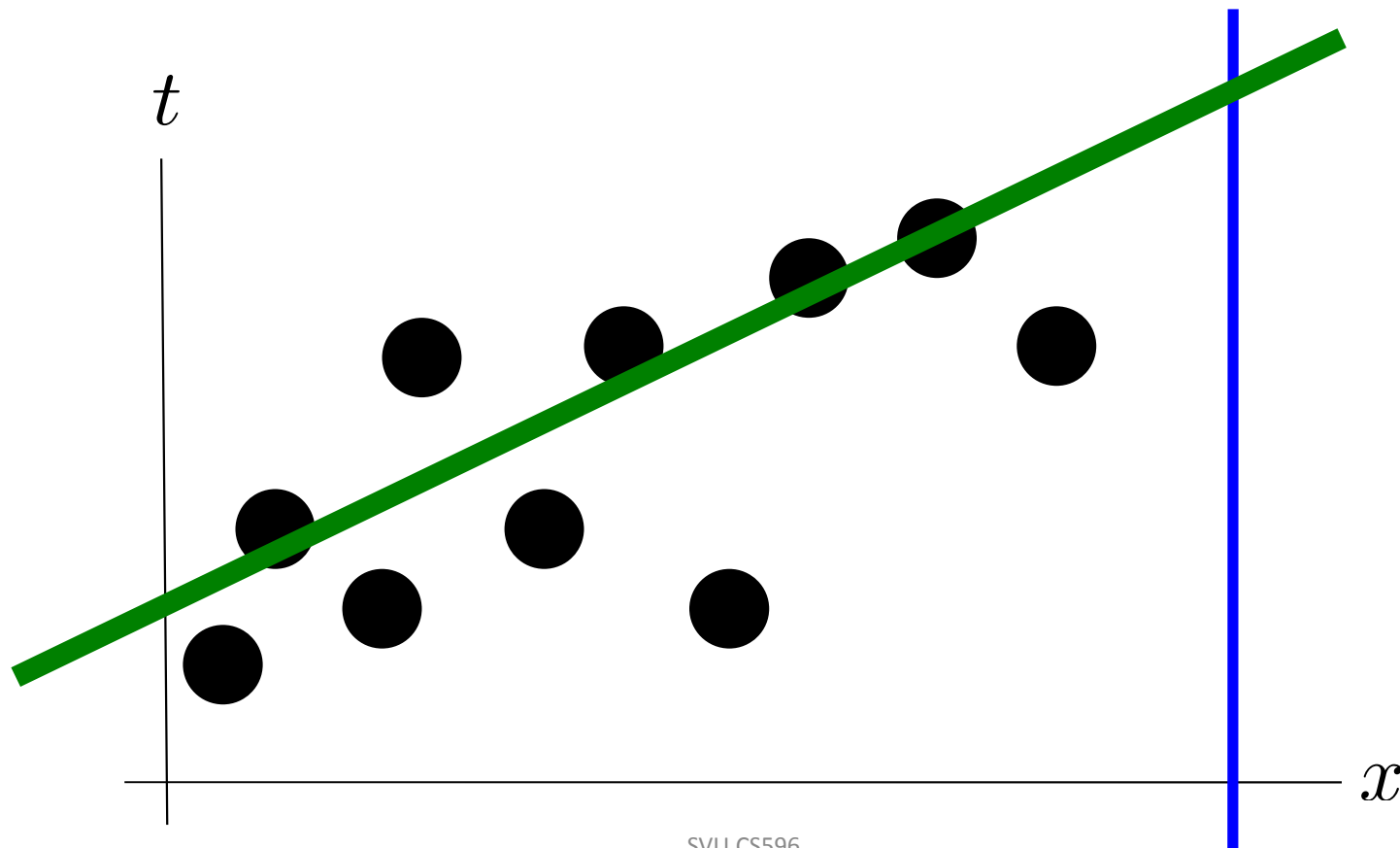
- Similar goals: Identify  $y(\mathbf{x}) = \mathbf{t}$ .
- What are the differences?
  - The form of the function,  $y$  (naturally).
  - Evaluation
    - Root Mean Squared Error
    - Absolute Value Error
    - Classification Error
    - Maximum Likelihood
  - Evaluation drives the optimization operation that learns the function,  $y$ .



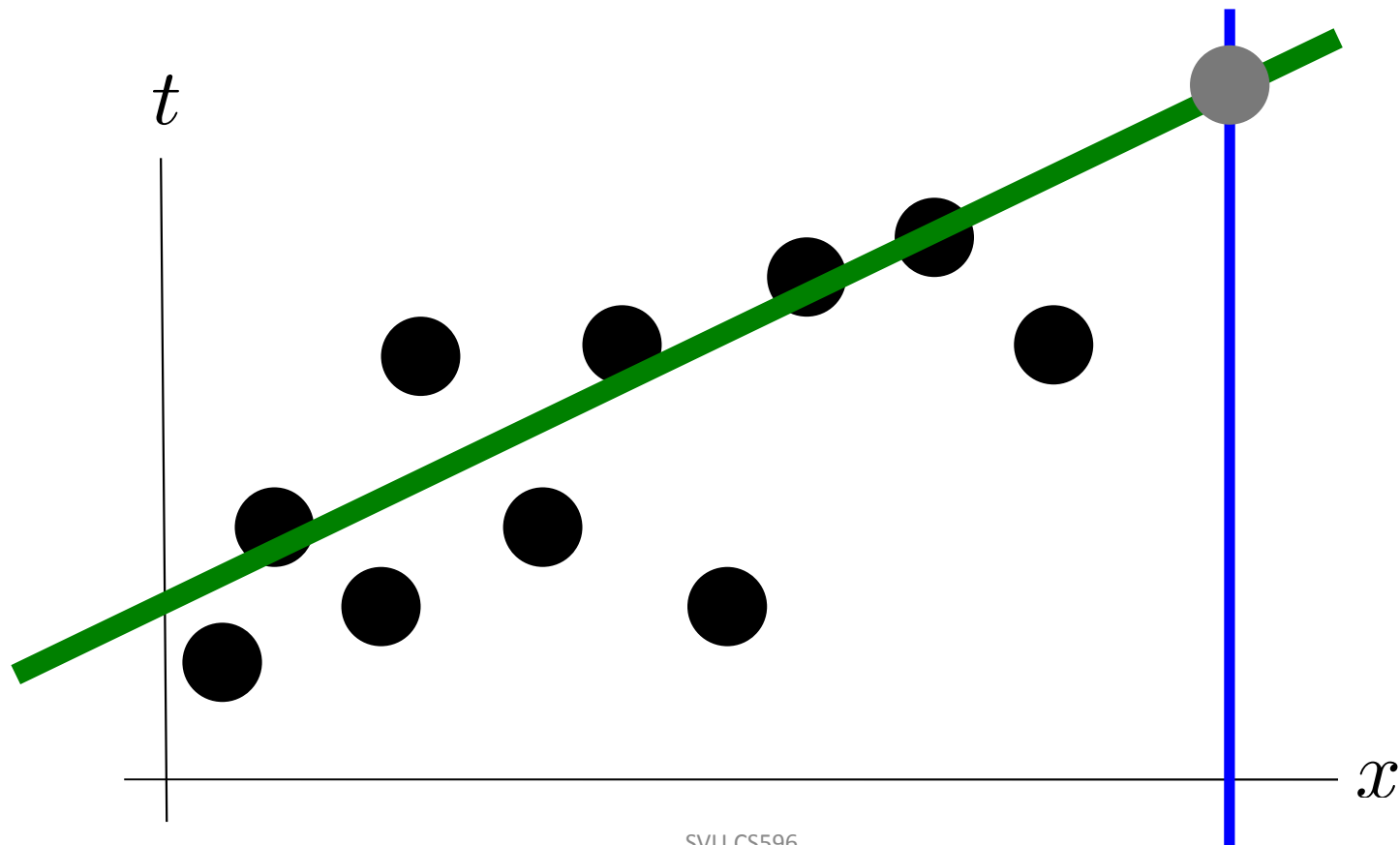
# Graphical Example of Regression



# Graphical Example of Regression



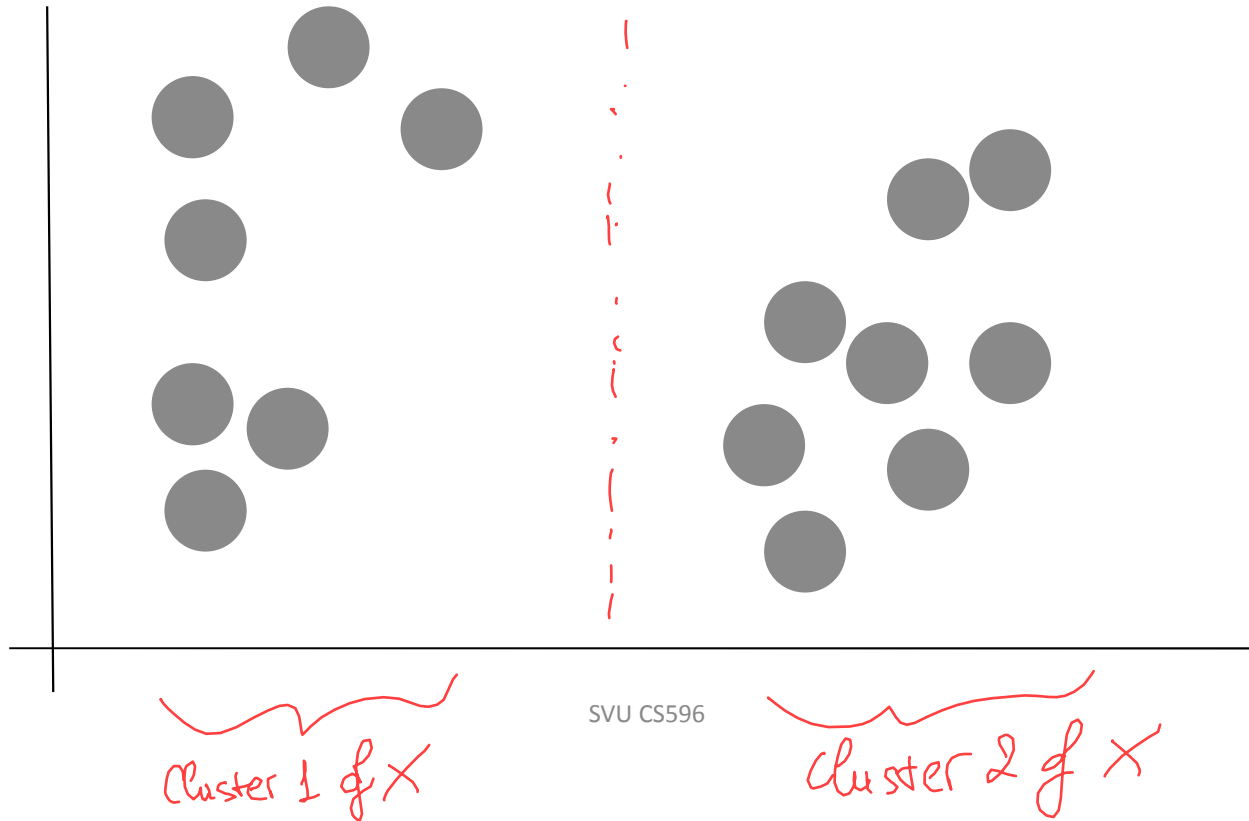
# Graphical Example of Regression



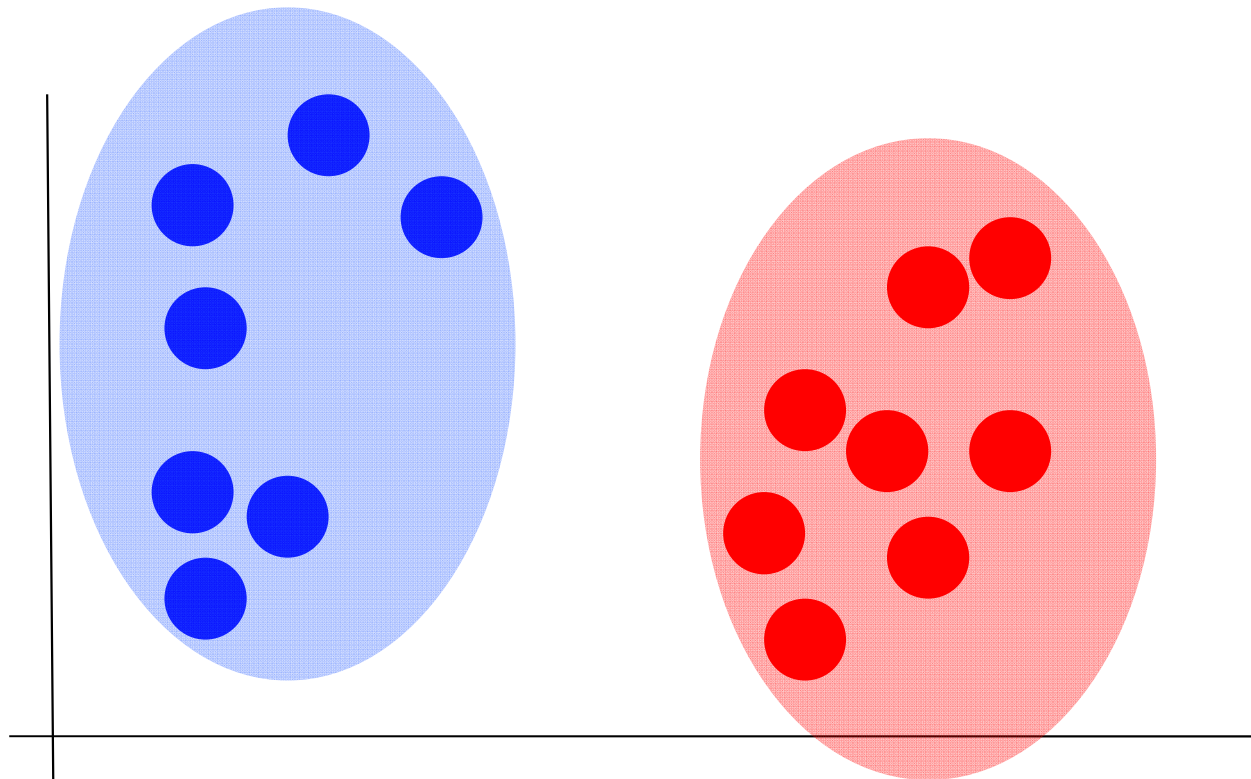
# Clustering

- Clustering is an **unsupervised** learning task.
  - There is no target value to shoot for.
- Identify groups of “similar” data points, that are “dissimilar” from others. *Extract similarity of the inputs*
- **Partition** the data into groups (clusters) that satisfy these constraints
  1. Points in the same cluster should be **similar**.
  2. Points in different clusters should be **dissimilar**.

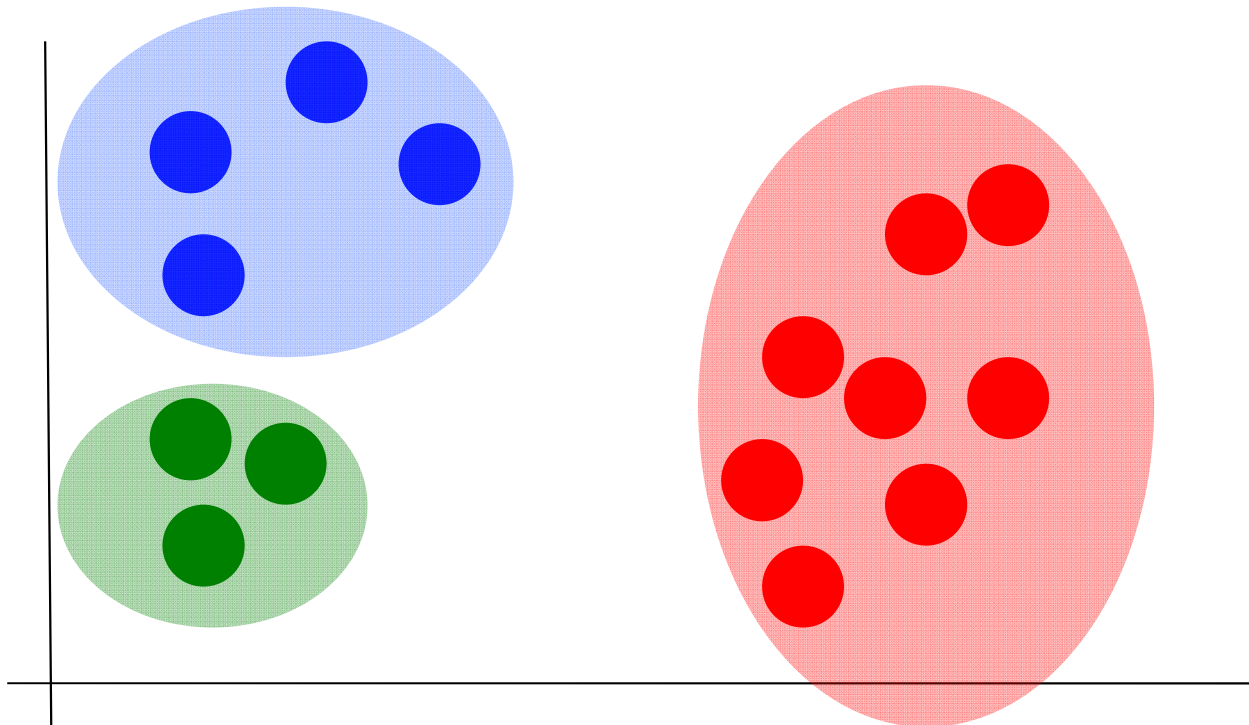
# Graphical Example of Clustering



# Graphical Example of Clustering



# Graphical Example of Clustering





# A Tutorial on the Python Programming Language

Moshe Goldstein



## Which of these languages do you know?

- C or C++
- Java
- Perl
- Scheme
- Fortran
- Python
- Matlab

# Popular Languages used in Machine Learning

- R
- Matlab
- Python
  - An interpreted script language
  - Interactive
  - Object oriented
  - Open source, fully free
  - Can also be used in other applications, e.g.,
    - Web service
    - Testing
    - Text processing

# Anaconda Installation

- Download at <http://continuum.io/downloads>
- Find the right package
- Follow the instructions

# Presentation Overview

- Running Python and Output
- Data Types
- Input and File I/O
- Control Flow
- Functions
- Then, Why Python in Scientific Computation?
- Binary distributions Scientific Python

# Hello World

- Open a terminal window and type “python”
- If on Windows open a Python IDE like IDLE
- At the prompt type ‘hello world!’

```
>>> 'hello world!'  
'hello world!'
```

# Python Overview

From *Learning Python, 2nd Edition*:

- Programs are composed of modules
- Modules contain statements
- Statements contain expressions
- Expressions create and process objects

# The Python Interpreter

- Python is an interpreted language
- The interpreter provides an interactive environment to play with the language
- Results of expressions are printed on the screen

```
>>> 3 + 7
10
>>> 3 < 15
True
>>> 'print me'
'print me'
>>> print 'print me'
print me
>>>
```

# The print Statement

- Elements separated by commas print with a space between them
- A comma at the end of the statement (`print 'hello',`) will not print a newline character

```
>>> print 'hello'
hello
>>> print 'hello', 'there'
hello there
```



# Documentation

The '#' starts a line comment

```
>>> 'this will print'  
'this will print'  
>>> #'this will not'  
>>>
```

# Variables

- Are not declared, just assigned
- The variable is created the first time you assign it a value
- Are references to objects
- Type information is with the object, not the reference
- Everything in Python is an object

# Everything is an object

- Everything means everything, including functions and classes (more on this later!)
- Data type is a property of the object and not of the variable

```
>>> x = 7
>>> x
7
>>> x = 'hello'
>>> x
'hello'
>>>
```

# Numbers: Integers

- Integer – the equivalent of a C long
- Long Integer – an unbounded integer value.

```
>>> 132224
132224
>>> 132323 ** 2
17509376329L
>>>
```

# Numbers: Floating Point

- `int(x)` converts `x` to an integer
- `float(x)` converts `x` to a floating point
- The interpreter shows a lot of digits

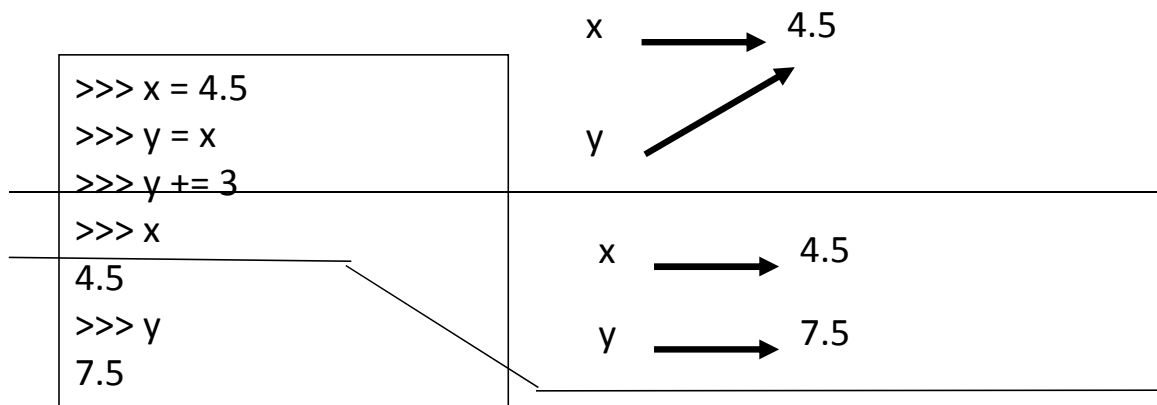
```
>>> 1.23232
1.2323200000000001
>>> print 1.23232
1.23232
>>> 1.3E7
13000000.0
>>> int(2.0)
2
>>> float(2)
2.0
```

# Numbers: Complex

- Built into Python
- Same operations are supported as integer and float

```
>>> x = 3 + 2j
>>> y = -1j
>>> x + y
(3+1j)
>>> x * y
(2-3j)
```

# Numbers are *immutable*



# String Literals

- Strings are *immutable*
- There is no char type like in C++ or Java
- + is overloaded to do concatenation

cannot be changed  
i.e. thread safe, no sync required

```
>>> x = 'hello'  
>>> x = x + ' there'  
>>> x  
'hello there'
```



# String Literals: Many Kinds

- Can use single or double quotes, and three double quotes for a multi-line string

```
>>> 'I am a string'
'I am a string'
>>> "So am I!"
'So am I!'
>>> s = """And me too!
though I am much longer
than the others :)"""
'And me too!\nthough I am much longer\nthan the others :)'
>>> print s
And me too!
though I am much longer
than the others :)'
```

# Substrings and Methods

```
>>> s = '012345'
>>> s[3]
'3'
>>> s[1:4]
'123'
>>> s[2:]
'2345'
>>> s[:4]
'0123'
>>> s[-2]
'4'
```

- **len**(String) – returns the number of characters in the String
- **str**(Object) – returns a String representation of the Object


```
>>> len(x)
6
>>> str(10.3)
'10.3'
```

# String Formatting

- Similar to C's printf
- <formatted string> % <elements to insert>
- Can usually just use %s for everything, it will convert the object to its String representation.

```
>>> "One, %d, three" % 2
'One, 2, three'
>>> "%d, two, %s" % (1,3)
'1, two, 3'
>>> "%s two %s" % (1, 'three')
'1 two three'
>>>
```

# Lists

- Ordered collection of data
- Data can be of different types
- Lists are *mutable*  *can be changed*
- Issues with shared references and mutability
- Same subset operations as Strings

```
>>> x = [1, 'hello', (3 + 2j)]
>>> x
[1, 'hello', (3+2j)]
>>> x[2]
(3+2j)
>>> x[0:2]
[1, 'hello']
```

*number of items*

# Lists: Modifying Content

- **`x[i] = a`** reassigns the *i*th element to the value *a*
- Since *x* and *y* point to the same list object, *both* are changed
- The method **`append`** also modifies the list

```
>>> x = [1,2,3]
>>> y = x
>>> x[1] = 15
>>> x
[1, 15, 3]
>>> y
[1, 15, 3]
>>> x.append(12)
>>> y
[1, 15, 3, 12]
```

# Lists: Modifying Contents

- The method **append** modifies the list and returns **None**
- List addition (+) returns a new list

```
>>> x = [1,2,3]
>>> y = x
>>> z = x.append(12)
>>> z == None
True
>>> y
[1, 2, 3, 12]
>>> x = x + [9,10]
>>> x
[1, 2, 3, 12, 9, 10]
>>> y
[1, 2, 3, 12]
>>>
```

# Tuples

unchanged

- Tuples are *immutable* versions of lists
- One strange point is the format to make a tuple with one element:  
' is needed to differentiate from the mathematical expression (2)

```
>>> x = (1,2,3)
>>> x[1:]
(2, 3)
>>> y = (2,)
>>> y
(2,)
>>>
```

# Dictionaries

- A set of key-value pairs
- Dictionaries are *mutable*

```
>>> d = {1 : 'hello', 'two' : 42, 'blah' : [1,2,3]}
>>> d
{1: 'hello', 'two': 42, 'blah': [1, 2, 3]}
>>> d['blah']
[1, 2, 3]
```



# Dictionaries: Add/Modify

- Entries can be changed by assigning to that entry

```
>>> d
{1: 'hello', 'two': 42, 'blah': [1, 2, 3]}
>>> d['two'] = 99
>>> d
{1: 'hello', 'two': 99, 'blah': [1, 2, 3]}
```

- Assigning to a key that does not exist adds an entry

```
>>> d[7] = 'new entry'
>>> d
{1: 'hello', 7: 'new entry', 'two': 99, 'blah': [1, 2, 3]}
```

# Dictionaries: Deleting Elements

- The **del** method deletes an element from a dictionary

```
>>> d
{1: 'hello', 2: 'there', 10: 'world'}
>>> del(d[2])
>>> d
{1: 'hello', 10: 'world'}
```

# Copying Dictionaries and Lists

- The built-in **list** function will copy a list
- The dictionary has a method called **copy**

List

```
>>> l1 = [1]
>>> l2 = list(l1)
>>> l1[0] = 22
>>> l1
[22]
>>> l2
[1]
```

Dictionary

```
>>> d = {1 : 10}
>>> d2 = d.copy()
>>> d[1] = 22
>>> d
{1: 22}
>>> d2
{1: 10}
```

# Data Type Summary

- Lists, Tuples, and Dictionaries can store any type (including other lists, tuples, and dictionaries!)
- Only lists and dictionaries are mutable
- All variables are references

# Data Type Summary

- Integers: 2323, 3234L
- Floating Point: 32.3, 3.1E2
- Complex: 3 + 2j, 1j
- Lists: l = [ 1,2,3]
- Tuples: t = (1,2,3)
- Dictionaries: d = {'hello' : 'there', 2 : 15}

# Input

- The **raw\_input**(string) method returns a line of user input as a string
- The parameter is used as a prompt
- The string can be converted by using the conversion methods **int**(string), **float**(string), etc.

# Input: Example

```
print "What's your name?"  
name = raw_input("> ")  
  
print "What year were you born?"  
birthyear = int(raw_input("> "))  
  
print "Hi %s! You are %d years old!" % (name, 2011 - birthyear)
```

```
~: python input.py  
What's your name?  
> Michael  
What year were you born?  
>1980  
Hi Michael! You are 31 years old!
```

# Files: Input

<code>inflobj = open('data', 'r')</code>	Open the file 'data' for input
<code>S = inflobj.read()</code>	Read whole file into one String
<code>S = inflobj.read(N)</code>	Reads N bytes ( $N \geq 1$ )
<code>L = inflobj.readlines()</code>	Returns a list of line strings



# Files: Output

<code>outflobj = open('data', 'w')</code>	Open the file 'data' for writing
<code>outflobj.write(S)</code>	Writes the string S to file
<code>outflobj.writelines(L)</code>	Writes each of the strings in list L to file
<code>outflobj.close()</code>	Closes the file

# Booleans

- 0 and None are false
- Everything else is true
- True and False are aliases for 1 and 0 respectively

# Boolean Expressions

- Compound boolean expressions short circuit
- and and or return one of the elements in the expression
- Note that when None is returned the interpreter does not print anything

```
>>> True and False
False
>>> False or True
True
>>> 7 and 14
14
>>> None and 2
>>> None or 2
2
```

# Moving to Files

- The interpreter is a good place to try out some code, but what you type is not reusable
- Python code files can be read into the interpreter using the **import** statement

## Moving to Files

- In order to be able to find a module called myscripts.py, the interpreter scans the list sys.path of directory names.
- The module must be in one of those directories.

```
>>> import sys
>>> sys.path
['C:\\Python26\\Lib\\idlelib', 'C:\\WINDOWS\\system32\\python26.zip',
'C:\\Python26\\DLLs', 'C:\\Python26\\lib', 'C:\\Python26\\lib\\plat-win',
'C:\\Python26\\lib\\lib-tk', 'C:\\Python26', 'C:\\Python26\\lib\\site-
packages']
>>> import myscripts
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    import myscripts.py
ImportError: No module named myscripts.py
```

# No Braces

- Python uses **indentation** instead of braces to determine the scope of expressions
- All lines must be indented the same amount to be part of the scope (or indented more if part of an inner scope)
- This **forces** the programmer to use proper indentation since the indenting is part of the program!

# If Statements

```
import math
x = 30
if x <= 15 :
    y = x + 15
elif x <= 30 :
    y = x + 30
else :
    y = x
print 'y = ',
print math.sin(y)
```

In file ifstatement.py

```
>>> import ifstatement
y = 0.999911860107
>>>
```

In interpreter

# While Loops

```
x = 1
while x < 10 :
    print x
    x = x + 1
```

In whileloop.py

```
>>> import whileloop
1
2
3
4
5
6
7
8
9
>>>
```

In interpreter



# Loop Control Statements

<b>break</b>	Jumps out of the closest enclosing loop
<b>continue</b>	Jumps to the top of the closest enclosing loop
<b>pass</b>	Does nothing, empty statement placeholder

# The Loop Else Clause

- The optional **else** clause runs only if the loop exits normally (not by break)

```
x = 1

while x < 3 :
    print x
    x = x + 1
else:
    print 'hello'
```

In whileelse.py

```
~: python whileelse.py
1
2
hello
```

Run from the command line

# The Loop Else Clause

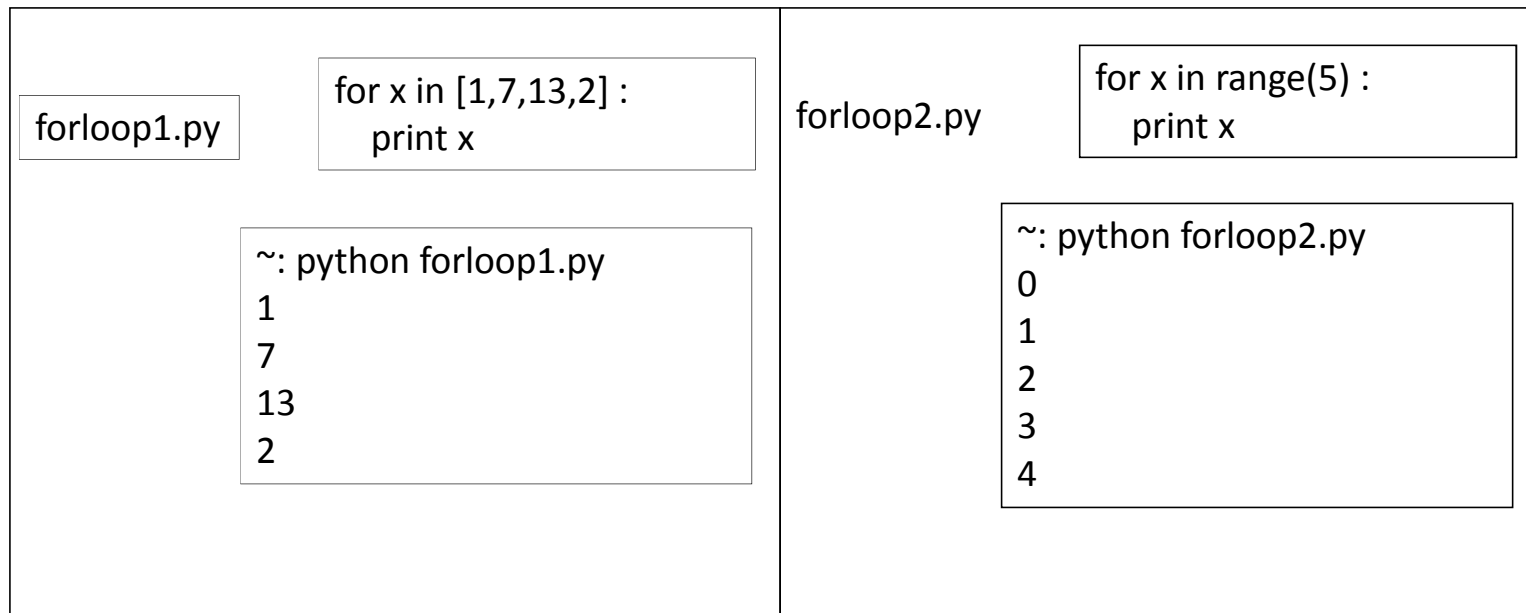
```
x = 1
while x < 5 :
    print x
    x = x + 1
    break
else :
    print 'i got here'
```

```
~: python whileelse2.py
1
```

whileelse2.py

# For Loops

- Similar to perl for loops, iterating through a list of values



range(N) generates a list of numbers [0,1, ..., n-1]

# For Loops

- **For** loops also may have the optional **else** clause

```
for x in range(5):  
    print x  
    break  
else :  
    print 'i got here'
```

```
~: python elseforloop.py  
1
```

elseforloop.py

# Function Basics

typo  
 $x > y$

```
def max(x,y):  
    if x < y:  
        return x  
    else:  
        return y
```

functionbasics.py

```
>>> import functionbasics  
>>> max(3,5)  
5  
>>> max('hello', 'there')  
'there'  
>>> max(3, 'hello')  
'hello'
```

# Functions are first class objects

- Can be assigned to a variable
- Can be passed as a parameter
- Can be returned from a function
- Functions are treated like any other variable in Python, the **def** statement simply assigns a function to a variable

# Function names are like any variable

- Functions are objects
- The same reference rules hold for them as for other objects

```
>>> x = 10
>>> x
10
>>> def x () :
...     print 'hello'
>>> x
<function x at 0x619f0>
>>> x()
hello
>>> x = 'blah'
>>> x
'blah'
```



# Functions as Parameters

```
def foo(f, a) :  
    return f(a)  
  
def bar(x) :  
    return x * x
```

```
>>> from funcasparam import *  
>>> foo(bar, 3)  
9
```

funcasparam.py

Note that the function **foo** takes two parameters and applies the first as a function with the second as its parameter

# Higher-Order Functions

**map(func,seq)** – for all i, applies func(seq[i]) and returns the corresponding sequence of the calculated results.

```
def double(x):  
    return 2*x
```

highorder.py

```
>>> from highorder import *  
>>> lst = range(10)  
>>> lst  
[0,1,2,3,4,5,6,7,8,9]  
>>> map(double,lst)  
[0,2,4,6,8,10,12,14,16,18]
```

# Higher-Order Functions

**filter(boolfunc,seq)** – returns a sequence containing all those items in seq for which boolfunc is True.

```
def even(x):  
    return ((x%2 == 0))
```

highorder.py

```
>>> from highorder import *  
>>> lst = range(10)  
>>> lst  
[0,1,2,3,4,5,6,7,8,9]  
>>> filter(even,lst)  
[0,2,4,6,8]
```

# Higher-Order Functions

**reduce(func,seq)** – applies func to the items of seq, from left to right, two-at-time, to reduce the seq to a single value.

```
def plus(x,y):  
    return (x + y)
```

```
>>> from highorder import *  
>>> lst = ['h','e','l','l','o']  
>>> reduce(plus,lst)  
'hello'
```

highorder.py

# Functions Inside Functions

- Since they are like any other object, you can have functions inside functions

```
def foo (x,y) :  
    def bar (z) :  
        return z * 2  
    return bar(x) + y
```

```
>>> from funcinfunc import *  
>>> foo(2,3)  
7
```

funcinfunc.py

# Functions Returning Functions

```
def foo (x) :  
    def bar(y) :  
        return x + y  
    return bar  
# main  
f = foo(3)  
print f  
print f(2)
```

```
~: python funcreturnfunc.py  
<function bar at 0x612b0>  
5
```

funcreturnfunc.py

# Parameters: Defaults

- Parameters can be assigned default values
- They are overridden if a parameter is given for them
- The type of the default doesn't limit the type of a parameter

```
>>> def foo(x = 3) :  
...     print x  
...  
>>> foo()  
3  
>>> foo(10)  
10  
>>> foo('hello')  
hello
```

# Parameters: Named

- Call by name
- Any positional arguments must come before named ones in a call

```
>>> def foo (a,b,c) :  
...     print a, b, c  
...  
>>> foo(c = 10, a = 2, b = 14)  
2 14 10  
>>> foo(3, c = 2, b = 19)  
3 19 2
```



# Anonymous Functions

- A lambda expression returns a function object
- The body can only be a simple expression, not complex statements

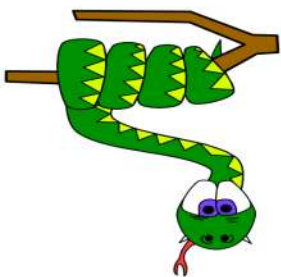
```
>>> f = lambda x,y : x + y
>>> f(2,3)
5
>>> lst = ['one', lambda x : x * x, 3]
>>> lst[1](4)
16
```

# Modules

- The highest level structure of Python
- Each file with the py suffix is a module
- Each module has its own namespace

# Modules: Imports

<code>import mymodule</code>	Brings all elements of mymodule in, but must refer to as mymodule.<elem>
<code>from mymodule import x</code>	Imports x from mymodule right into this namespace
<code>from mymodule import *</code>	Imports all elements of mymodule into this namespace



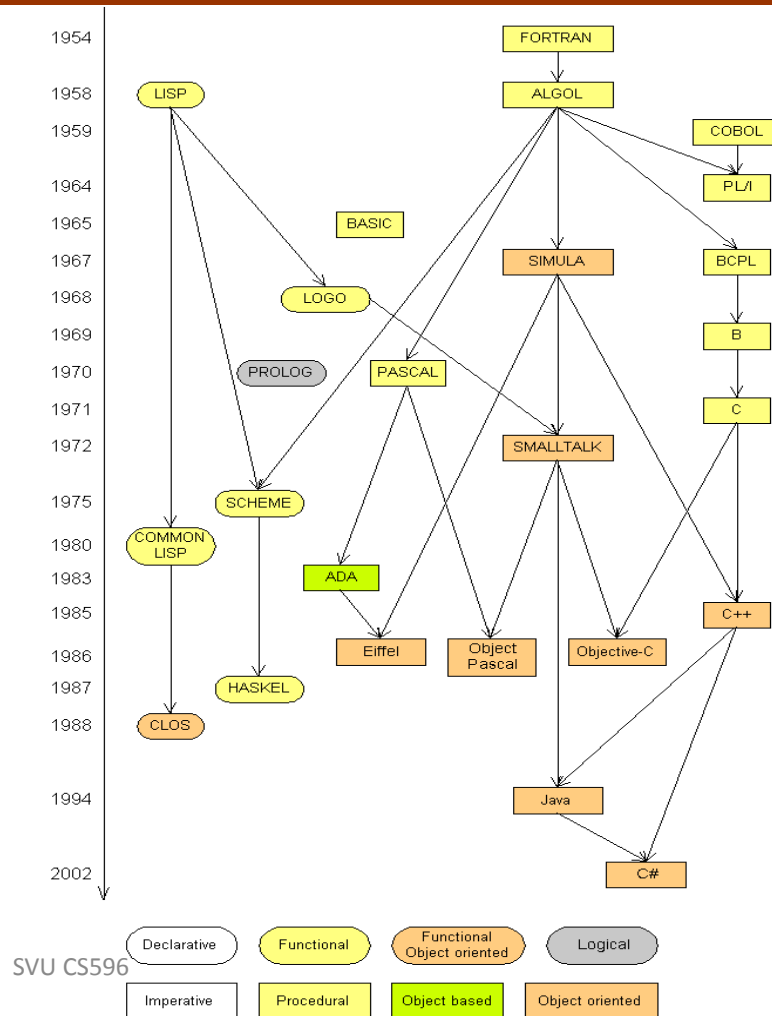
# Introduction to Programming with Python

Marty Stepp (stepp@cs.washington.edu)  
Lecturer, Computer Science & Engineering  
University of Washington

# Languages

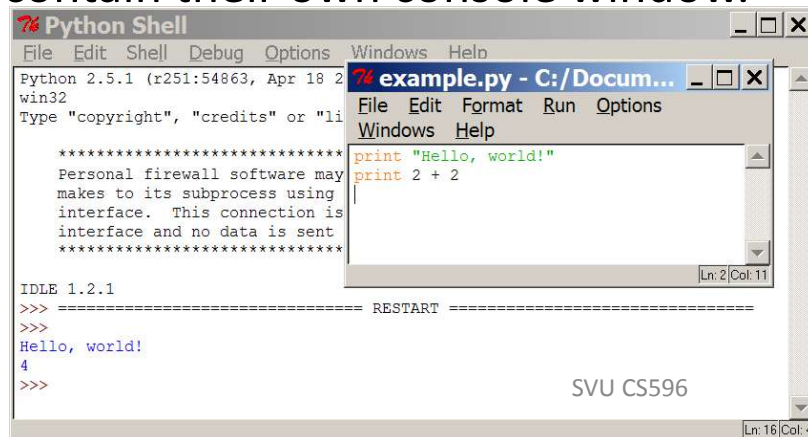
- Some influential ones:

- FORTRAN
  - science / engineering
- COBOL
  - business data
- LISP
  - logic and AI
- BASIC
  - a simple language



# Programming basics

- **code** or **source code**: The sequence of instructions in a program.
- **syntax**: The set of legal structures and commands that can be used in a particular programming language.
- **output**: The messages printed to the user by a program.
- **console**: The text box onto which output is printed.
  - Some source code editors pop up the console as an external window, and others contain their own console window.

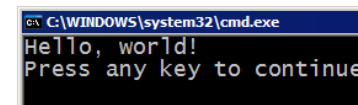


The screenshot shows a 'Python Shell' window with a menu bar (File, Edit, Shell, Debug, Options, Windows, Help). The main text area contains the following text:

```
Python 2.5.1 (r251:54863, Apr 18 2006) on win32
Type "copyright", "credits" or "license()" for more

>>>
>>> print "Hello, world!"
Hello, world!
>>> print 2 + 2
4
>>>
```

At the bottom right of the window, the text 'SVU CS596' is displayed. The status bar at the bottom indicates 'Ln: 16/Col: 4'.

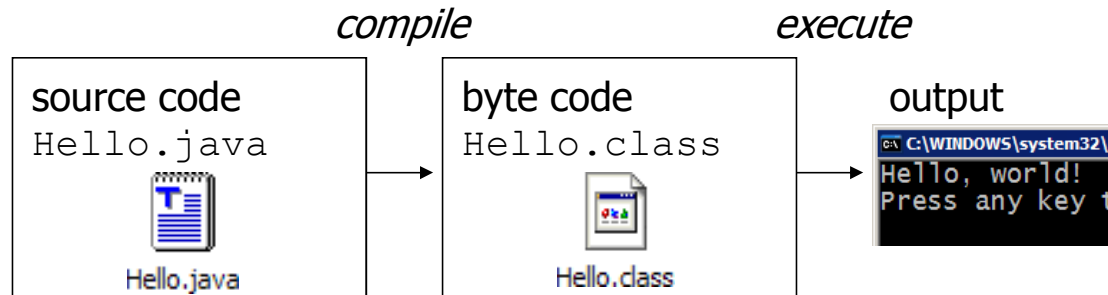


The screenshot shows a Windows command prompt window titled 'C:\WINDOWS\system32\cmd.exe'. The output displayed is:

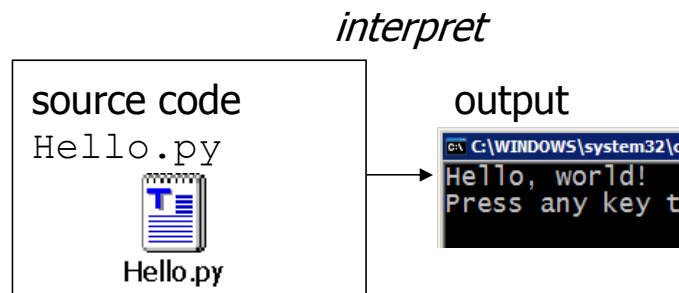
```
Hello, world!
Press any key to continue
```

# Compiling and interpreting

- Many languages require you to *compile* (translate) your program into a form that the machine understands.



- Python is instead directly *interpreted* into machine instructions.



# Expressions

- **expression:** A data value or set of operations to compute a value.

Examples:  $1 + 4 * 3$   
42

- Arithmetic operators we will use:

$+$	$-$	$*$	$/$	addition, subtraction/negation, multiplication, division
$\%$				modulus, a.k.a. remainder
$**$				exponentiation

- **precedence:** Order in which operations are computed.

- $*$   $/$   $\%$   $**$  have a higher precedence than  $+$   $-$

$1 + 3 * 4$  is 13

- Parentheses can be used to force a certain order of evaluation.

$(1 + 3) * 4$  is 16



# Real numbers

- Python can also manipulate real numbers.
  - Examples: `6.022`    `-15.9997`    `42.0`    `2.143e17`
- The operators `+` `-` `*` `/` `%` `**` `()` all work for real numbers.
  - The `/` produces an exact answer: `15.0 / 2.0` is **`7.5`**
  - The same rules of precedence also apply to real numbers:  
Evaluate `()` before `*` `/` `%` before `+` `-`
- When integers and reals are mixed, the result is a real number.
  - Example: `1 / 2.0` is `0.5`
  - The conversion occurs on a per-operator basis.

$$\begin{array}{r} 7 / 3 * 1.2 + 3 / 2 \\ \underline{2} * 1.2 + 3 / 2 \\ 2.4 + 3 / 2 \\ \underline{2.4 + 1} \\ 3.4 \end{array}$$

# Math commands

- Python has useful [commands](#) for performing calculations.

Command name	Description
<code>abs(<b>value</b>)</code>	absolute value
<code>ceil(<b>value</b>)</code>	rounds up
<code>cos(<b>value</b>)</code>	cosine, in radians
<code>floor(<b>value</b>)</code>	rounds down
<code>log(<b>value</b>)</code>	logarithm, base e
<code>log10(<b>value</b>)</code>	logarithm, base 10
<code>max(<b>value1</b>, <b>value2</b>)</code>	larger of two values
<code>min(<b>value1</b>, <b>value2</b>)</code>	smaller of two values
<code>round(<b>value</b>)</code>	nearest whole number
<code>sin(<b>value</b>)</code>	sine, in radians
<code>sqrt(<b>value</b>)</code>	square root

Constant	Description
e	2.7182818...
pi	3.1415926...

- To use many of these commands, you must write the following at the top of your Python program:

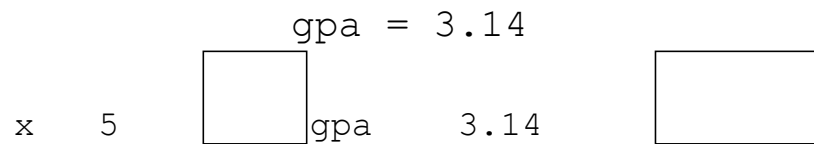
```
from math import *
```

# Variables

- **variable:** A named piece of memory that can store a value.
  - Usage:
    - Compute an expression's result,
    - store that result into a variable,
    - and use that variable later in the program.
- **assignment statement:** Stores a value into a variable.
  - Syntax:

***name*** = ***value***

- Examples: `x = 5`



- A variable that has been given a value can be used in expressions.  
`x + 4` is 9

# print

- `print` : Produces text output on the console.

- Syntax:

```
print "Message"
```

```
print Expression
```

- Prints the given text message or expression value on the console, and moves the cursor down to the next line.

```
print Item1, Item2, ..., ItemN
```

- Prints several messages and/or expressions on the same line.

- Examples:

```
print "Hello, world!"
```

```
age = 45
```

```
print "You have", 65 - age, "years until retirement"
```

Output:

```
Hello, world!
```

```
You have 20 years until retirement
```

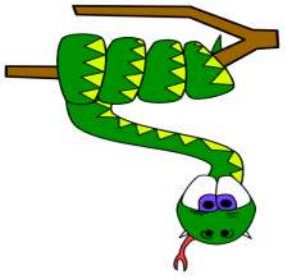
# input

- `input` : Reads a number from user input.
  - You can assign (store) the result of `input` into a variable.
  - Example:

```
age = input("How old are you? ")
print "Your age is", age
print "You have", 65 - age, "years until retirement"
```

Output:

```
How old are you? 53
Your age is 53
You have 12 years until retirement
```



Repetition (loops)  
and Selection (if/else)

# The `for` loop

- **`for` loop**: Repeats a set of statements over a group of values.

- Syntax:

```
for variableName in groupOfValues:  
    statements
```

- We indent the statements to be repeated with tabs or spaces.
  - ***variableName*** gives a name to each value, so you can refer to it in the ***statements***.
  - ***groupOfValues*** can be a range of integers, specified with the `range` function.
- Example:

```
for x in range(1, 6):  
    print x, "squared is", x * x
```

Output:

```
1 squared is 1  
2 squared is 4  
3 squared is 9  
4 squared is 16  
5 squared is 25
```

# range

- The `range` function specifies a range of integers:

- `range(start, stop)` - the integers between ***start*** (inclusive) and ***stop*** (exclusive)

- It can also accept a third value specifying the change between values.

- `range(start, stop, step)` - the integers between ***start*** (inclusive) and ***stop*** (exclusive) by ***step***

- Example:

```
for x in range(5, 0, -1):  
    print x  
print "Blastoff!"
```

Output:

```
5  
4  
3  
2  
1  
Blastoff!
```



# Cumulative loops

- Some loops incrementally compute a value that is initialized outside the loop. This is sometimes called a *cumulative sum*.

```
sum = 0
for i in range(1, 11):
    sum = sum + (i * i)
print "sum of first 10 squares is", sum
```

Output:  
sum of first 10 squares is 385

- **Exercise:** Write a Python program that computes the factorial of an integer.

# if

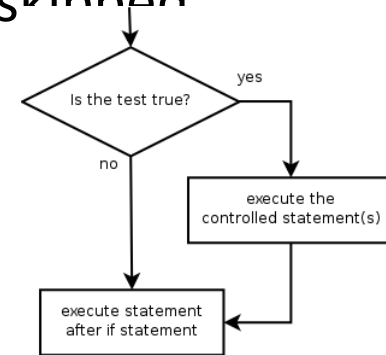
- **if statement:** Executes a group of statements only if a certain condition is true. Otherwise, the statements are *skipped*

- Syntax:

```
if condition:  
    statements
```

- Example:

```
gpa = 3.4  
if gpa > 2.0:  
    print "Your application is accepted."
```



# if/else

- **if/else statement:** Executes one block of statements if a certain condition is True, and a second block of statements if it is False.

- Syntax:

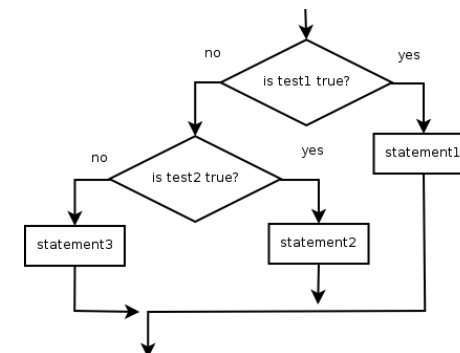
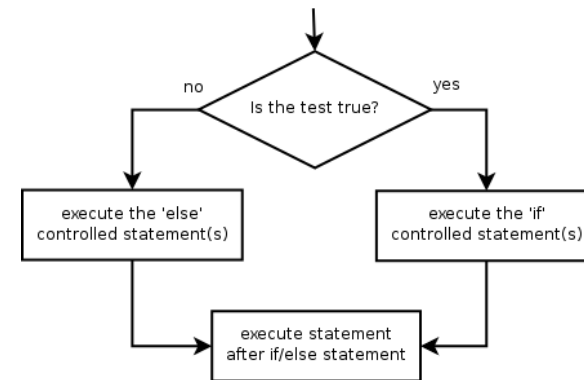
```
if condition:  
    statements  
else:  
    statements
```

- Example:

```
gpa = 1.4  
if gpa > 2.0:  
    print "Welcome to Mars University!"  
else:  
    print "Your application is denied."
```

- Multiple conditions can be chained with `elif` ("else if"):

```
if condition:  
    statements  
elif condition:  
    statements  
else:  
    statements
```



# while

- **while loop:** Executes a group of statements as long as a condition is True.
  - good for *indefinite loops* (repeat an unknown number of times)

- Syntax:

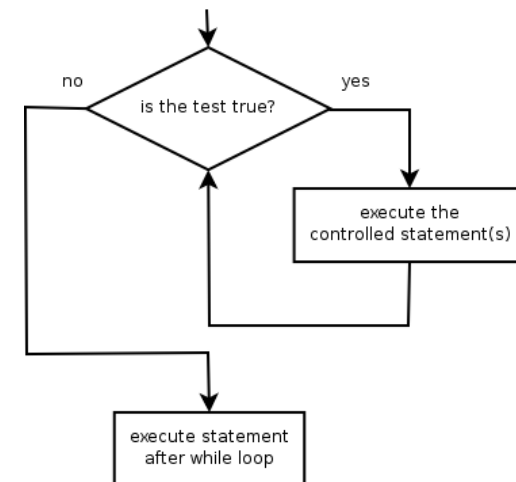
```
while condition:  
    statements
```

- Example:

```
number = 1  
while number < 200:  
    print number,  
    number = number * 2
```

- Output:

1 2 4 8 16 32 64 128



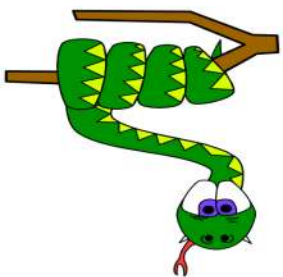
# Logic

- Many logical expressions use *relational operators*:

Operator	Meaning	Example	Result
==	equals	<code>1 + 1 == 2</code>	True
!=	does not equal	<code>3.2 != 2.5</code>	True
<	less than	<code>10 &lt; 5</code>	False
>	greater than	<code>10 &gt; 5</code>	True
<=	less than or equal to	<code>126 &lt;= 100</code>	False
>=	greater than or equal to	<code>5.0 &gt;= 5.0</code>	True

- Logical expressions can be combined with *logical operators*:

Operator	Example	Result
and	<code>9 != 6 and 2 &lt; 3</code>	True
or	<code>2 == 3 or -1 &lt; 5</code>	True
not	<code>not 7 &gt; 0</code>	False



# Text and File Processing

# Strings

- **string:** A sequence of text characters in a program.
  - Strings start and end with quotation mark " or apostrophe ' characters.
  - Examples:  
`"hello"`  
`"This is a string"`  
`"This, too, is a string. It can be very long!"`
- A string may not span across multiple lines or contain a " character.  
`"This is not  
a legal String."`  
`"This is not a "legal" String either."`
- A string can represent characters by preceding them with a backslash.
  - `\t` tab character
  - `\n` new line character
  - `\"` quotation mark character
  - `\\` backslash character
- Example: `"Hello\tthere\nHow are you?"`

# Indexes

- Characters in a string are numbered with *indexes* starting at 0:

- Example:

```
name = "P. Diddy"
```

index	0	1	2	3	4	5	6	7
character	P	.		D	i	d	d	y

- Accessing an individual character of a string:

***variableName* [ *index* ]**

- Example:

```
print name, "starts with", name[0]
```

Output:

```
P. Diddy starts with P
```



# String properties

- `len(string)` - number of characters in a string  
(including spaces)
- `str.lower(string)` - lowercase version of a string
- `str.upper(string)` - uppercase version of a string
- Example:  

```
name = "Martin Douglas Stepp"  
length = len(name)  
big_name = str.upper(name)  
print big_name, "has", length, "characters"
```

Output:

```
MARTIN DOUGLAS STEPP has 20 characters
```

# raw\_input

- `raw_input` : Reads a string of text from user input.

- Example:

```
name = raw_input("Howdy, pardner. What's yer  
name? ")  
print name, "... what a silly name!"
```

Output:

```
Howdy, pardner. What's yer name? Paris Hilton  
Paris Hilton ... what a silly name!
```

# Text processing

- **text processing:** Examining, editing, formatting text.
  - often uses loops that examine the characters of a string one by one
- A `for` loop can examine each character in a string in sequence.
- Example:

```
for c in "booyah":  
    print c
```

Output:

```
b  
o  
o  
y  
a  
h
```

# Strings and numbers

- `ord(text)` - converts a string into a number.
  - Example: `ord("a")` is 97, `ord("b")` is 98, ...
  - Characters map to numbers using standardized mappings such as *ASCII* and *Unicode*.
- `chr(number)` - converts a number into a string.
  - Example: `chr(99)` is "c"

# File processing

- Many programs handle data, which often comes from files.
- Reading the entire contents of a file:

```
variableName = open ("filename") .read()
```

Example:

```
file_text = open("bankaccount.txt") .read()
```

# Line-by-line processing

- Reading a file line-by-line:

```
for line in open("filename").readlines():  
    statements
```

Example:

```
count = 0  
for line in open("bankaccount.txt").readlines():  
    count = count + 1  
print "The file contains", count, "lines."
```

# Next Week

- Math primer
- Linear regression
- Python practice