MSML 605
Introduction

Administration

- Course webpage
  - Get homework assignments
    http://www.cs.umd.edu/~nayeem/courses/MSML605/
  - Syllabus
  - Other documents
- Piazza
  - Ask questions
    - Do not post solutions
    - Do not ask if your answer or approach is correct
  - Discuss issues
  - Public versus Private
- ELMS
  - Submit homework / assignments
  - See grades.

Administration (contd.)

- References
  - There is no specific textbook for this class.
  - We will be posting links to any references covered in class.
- Homework / Assignments
  - Regular homework and programming assignment.
  - Late date: 20% off your actual grade. (one get-out-of-jail-free card).

Components of the course

- Quizzes:

"are you with us?" not worth many points but useful finger on the pulse (for you and me)

- Tests :

"what have you learned?" Important checkpoints!!

- Programming assignments :

"can you implement it?"

Administration (contd.)

- Exams
  - One midterm : April 9 in lecture.
  - Final exam: May 14 in lecture.

- Grading
  - Quiz: 1% for each quiz
  - Midterm: 20%
  - Assignments: 30%
  - Final: 30-35%

- Academic integrity

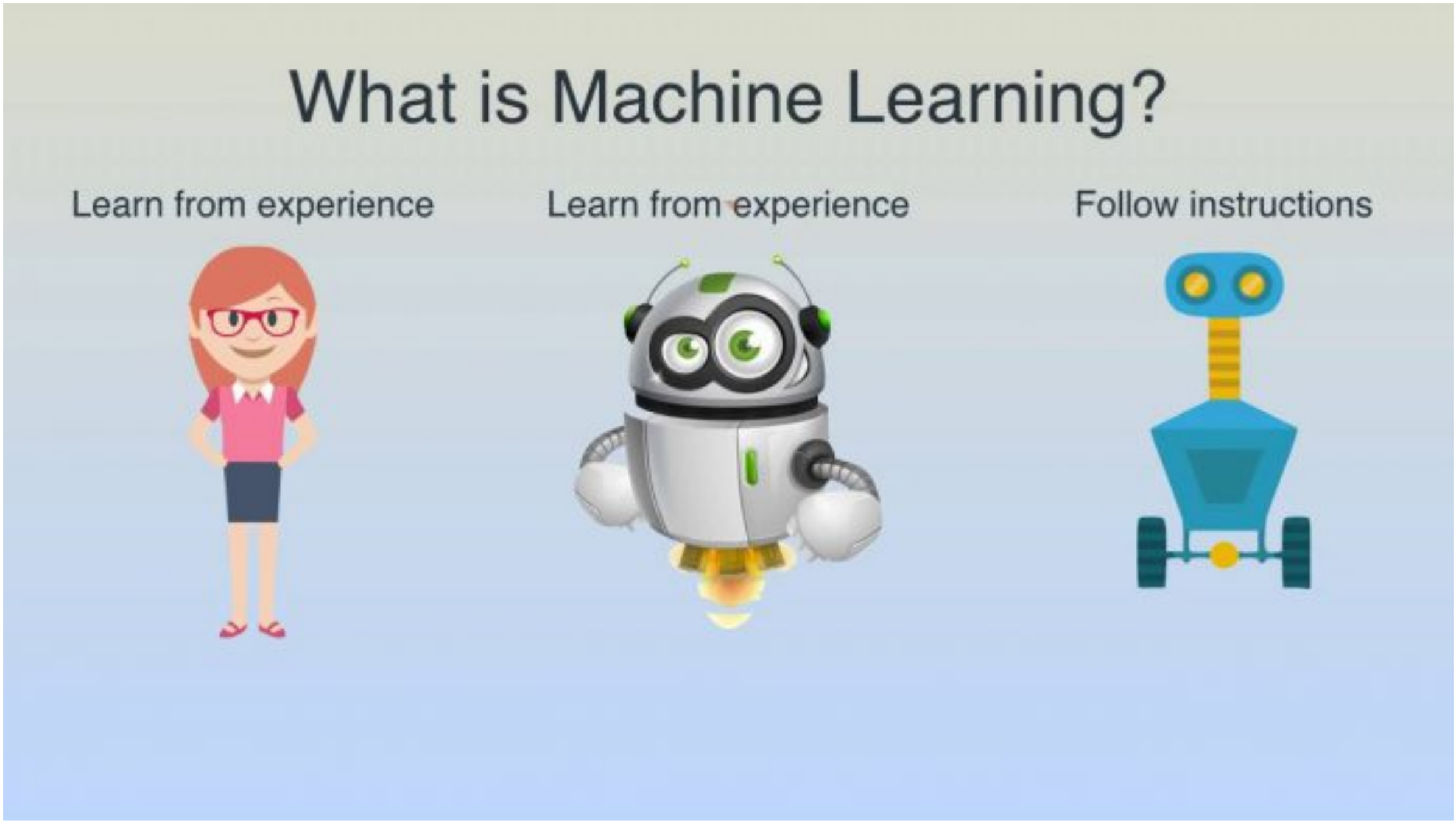
Topics (tentative)

- Introduction
- Python programming
- Numpy, Scipy
- Matplotlib
- Pandas
- Stats library
- Github
- Database connectivity
- Tensorflow
- Hardware for Machine learning

Introduction to Machine Learning

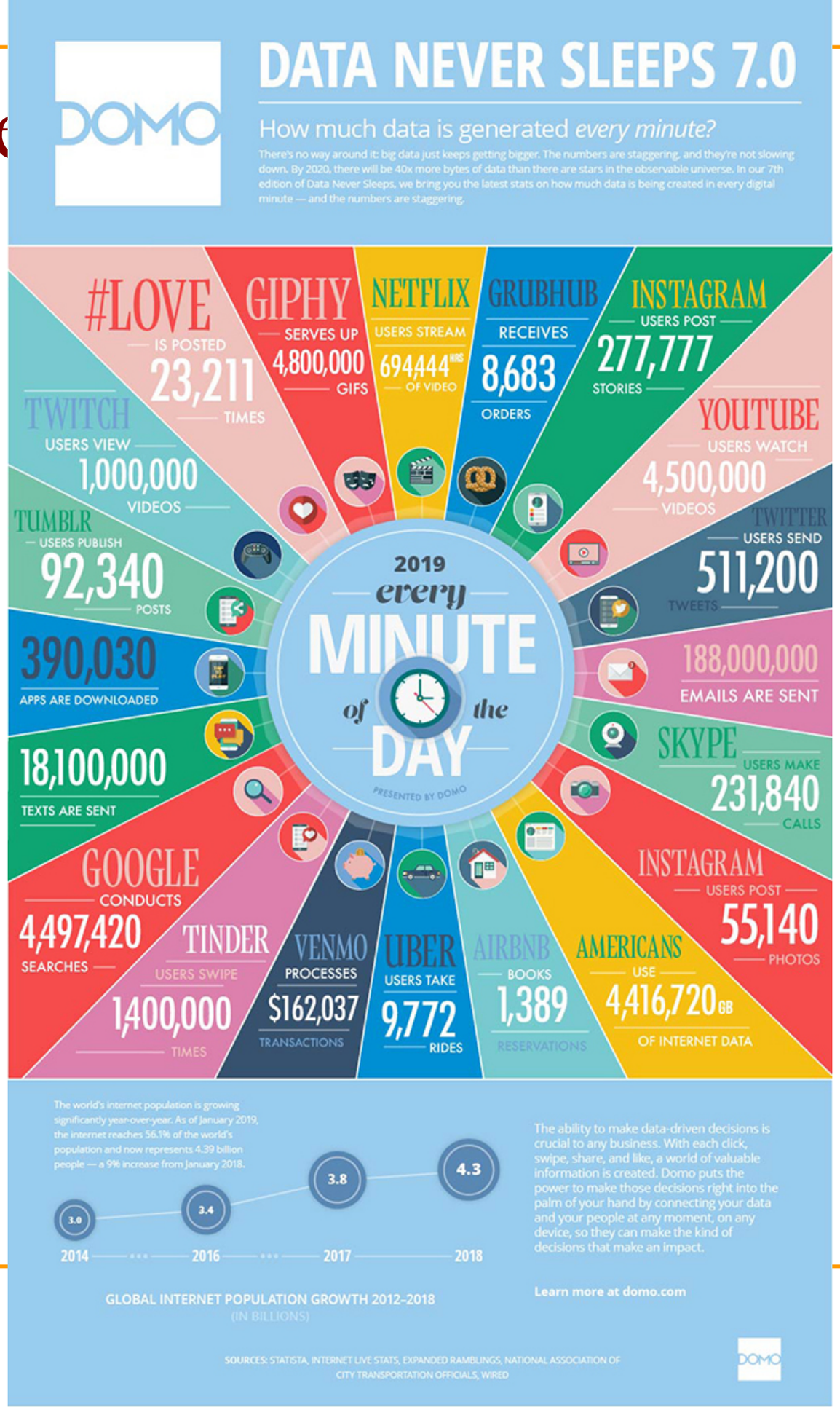
source: https://xkcd.com/1838/

Introduction



source: http://bigdata.black/wp-content/uploads/2016/05/whatis-machine-learning-0-696x392.jpg

Data Gene



https://www.domo.com/learn/data-never-sleeps-5?aid=ogsm072517_1&sf100871281=1

Data Generated

IBM

IBM Consumer Products Industry Blog

Industry Insights

2.5 quintillion bytes of data created every day. How does CPG & Retail manage it?

2.5 quintillions of data is generated per day

1 quintillion = 1,000,000,000,000 Million
= 1 Billion Billion

Scenarios

- Banks are building up pictures of how people spend their money,
- hospitals are recording what treatment patients are on for which ailments and how they respond.
- Engine monitoring systems are recording information about the engine.

The challenge is to do something useful with this data

Scenarios

- Enormous amount of biological data is available today, such as gene expression, protein transcription data and phylogenetic trees relating species to each other, etc.

- Around terabyte of data is collected every night in the form of Data collected from telescopes around the world.

- Medical science stores outcomes of medical tests from measurements such as MRI scans and simple blood tests.

The explosion of stored data is well known; the challenge is to do something useful with it.

Machine Learning Scenarios

- If the bank's computers can learn about spending patterns, can they detect credit card fraud quickly?
- If hospitals share data, then can treatments that don't work as well as expected be identified quickly?
- Can an intelligent car give you early warnings of problems, so that you don't end up stranded?

ML Interaction

ML Interaction

- You have already interacted with machine learning algorithms at some time.

- Spam filters
- Voice recognition
- Computer games
- Automatic license plate recognition on toll roads
- Recommendations by Amazon and Netflix

Different ML Algorithms

- Supervised Learning: Learning from exemplars
- Unsupervised Learning: labels are not provided
- Reinforcement Learning:Somewhere between supervised and unsupervised learning.

Supervised Learning - Classification

Training Data

cat

cat

cat

.

.

cat

dog

dog

dog

.

.

dog

Test Image

?

House Prices - Regression

Price (in 1000$)	Area(sq. ft.)	# Bathrooms	# Bedrooms
220	1600	2.5	3
y_i 180	1400	1.5	3
350	2100	3.5	4
…	…	…	…
….	….	…	…
500	2400	4	5

$x^{(i)}$

1400
1.5
3

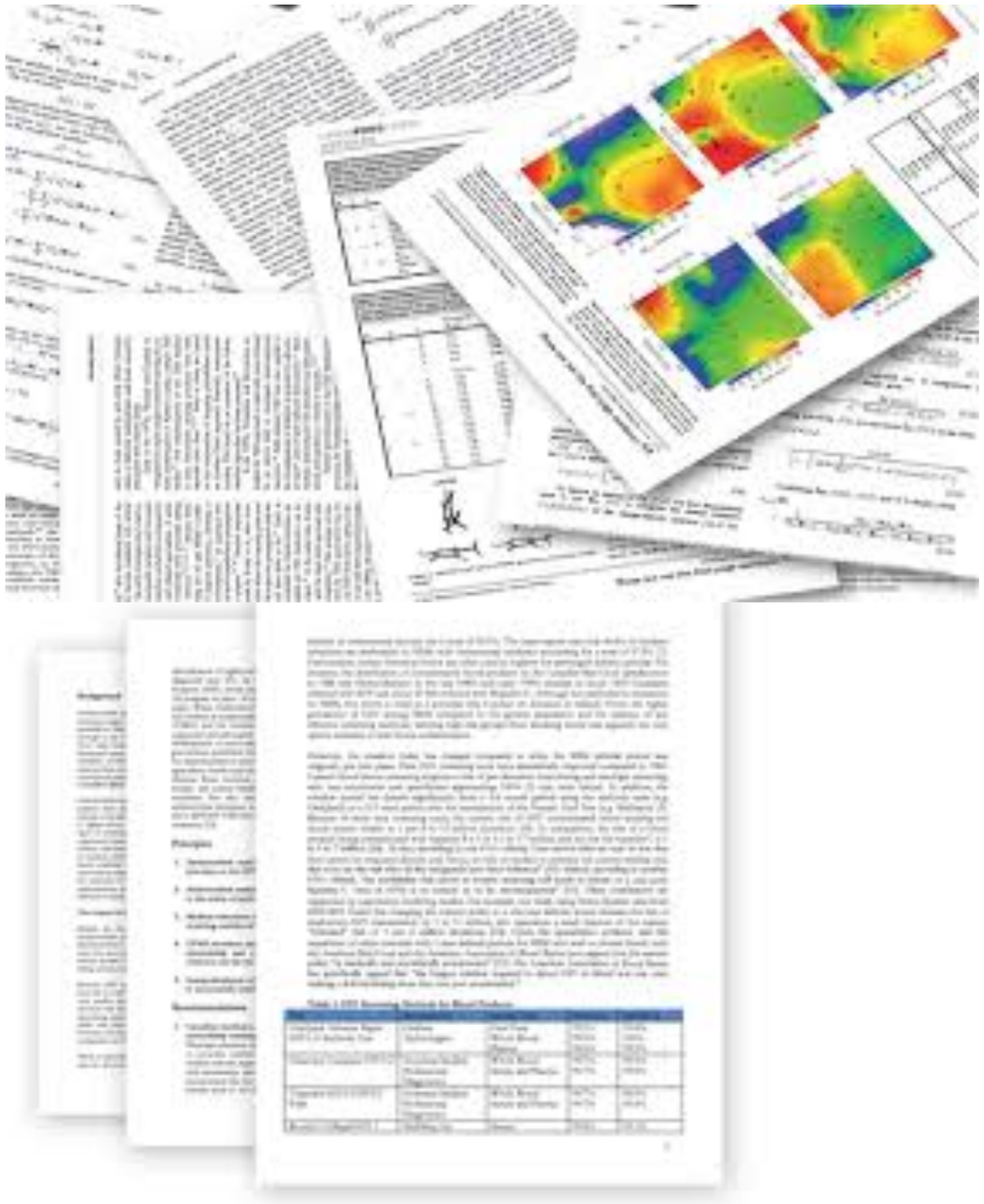
x^{test}

1850
2.5
3

y ?

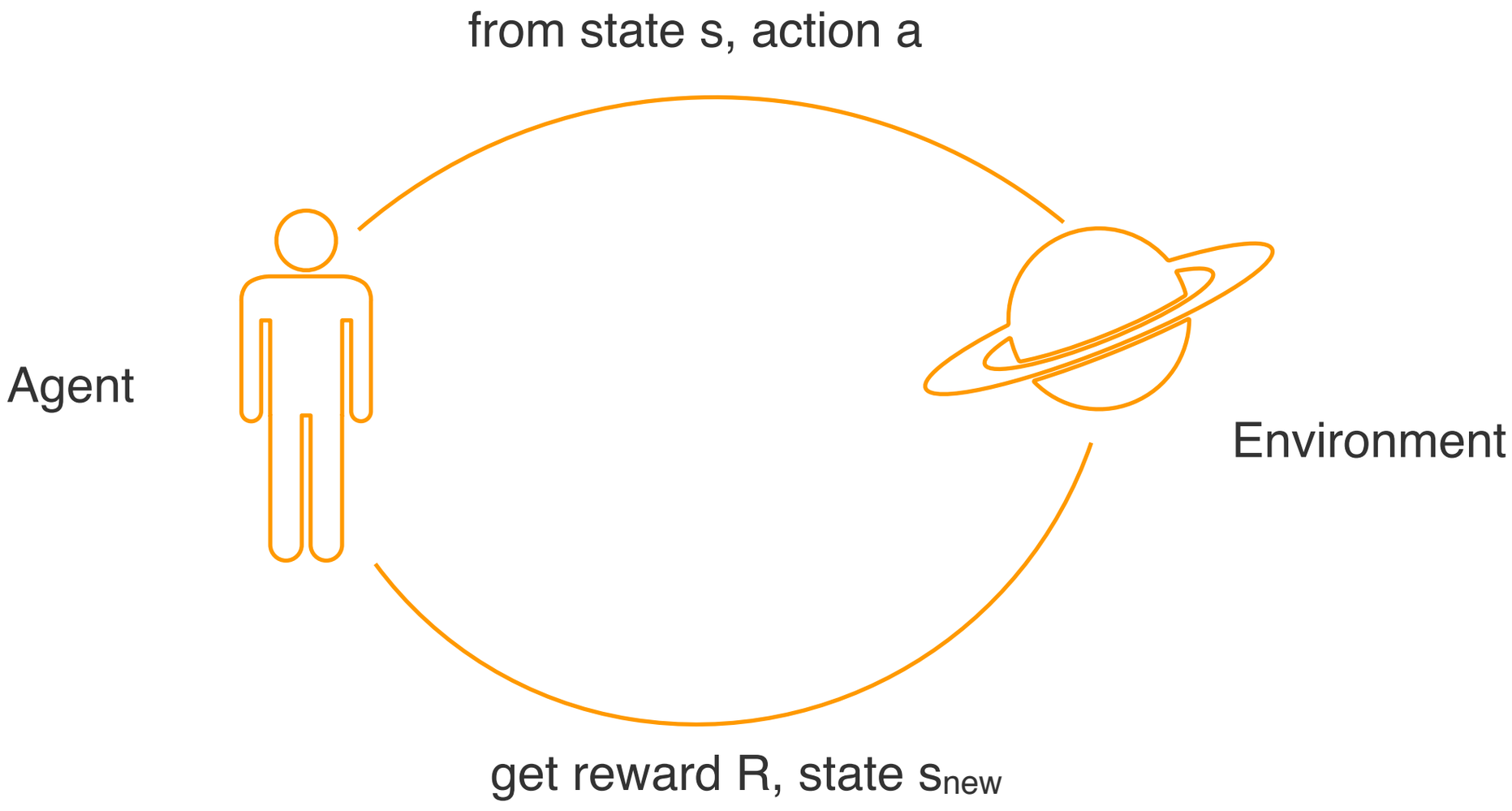
Unsupervised Learning

Training Data



Test Document

Reinforcement Learning

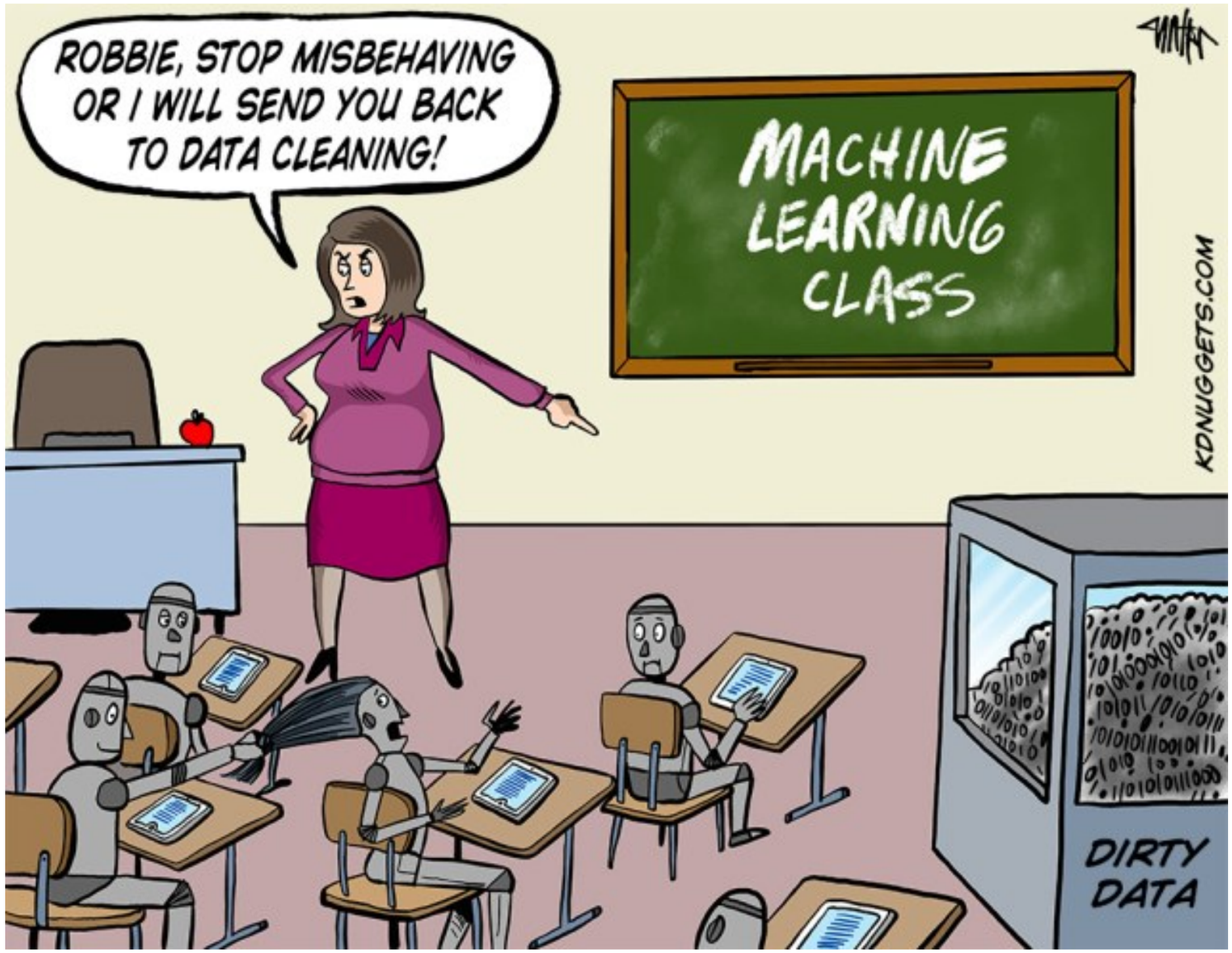


ML Approach

- Collect data

ML Approach

- Clean that data - prepare it for analysis



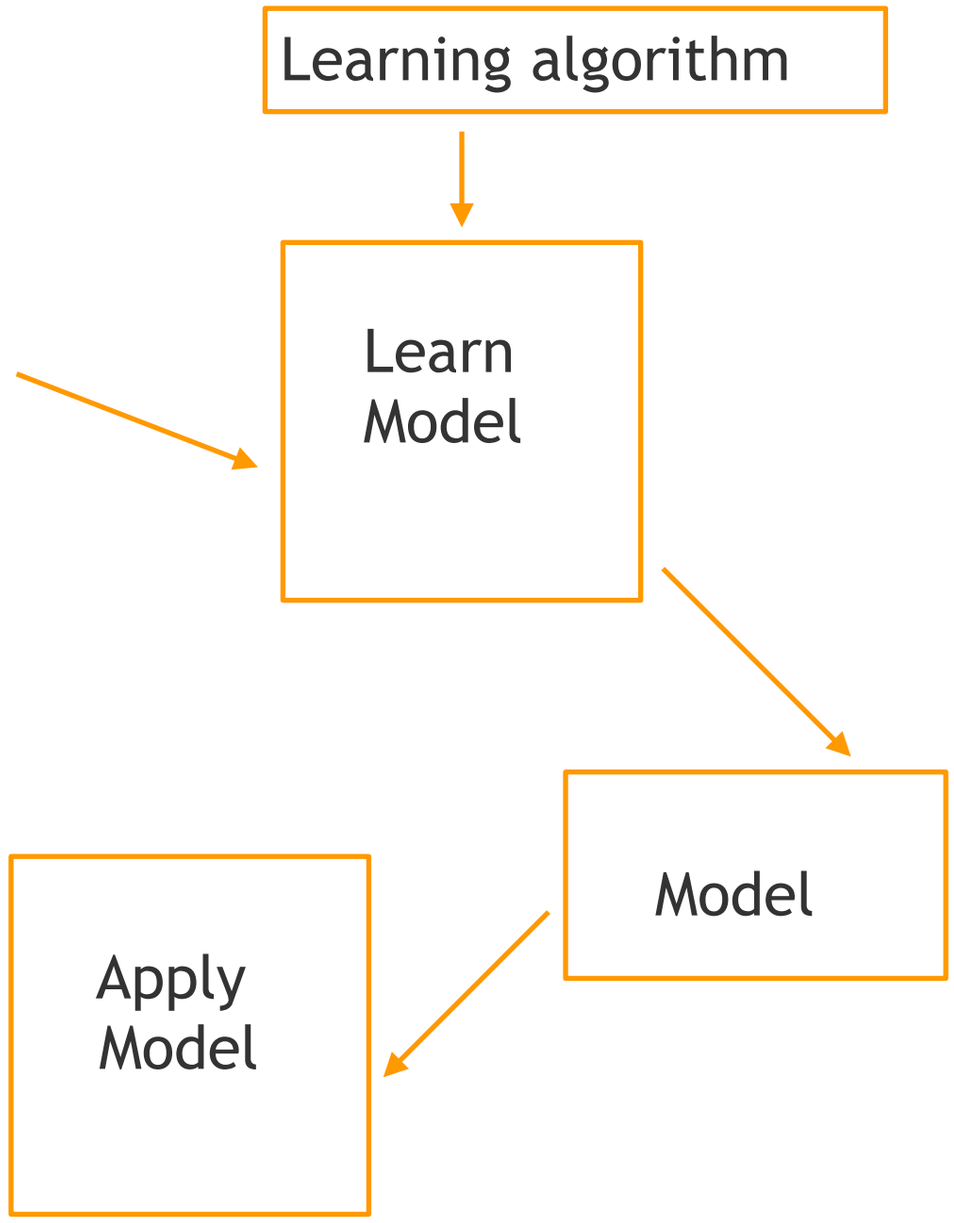
ML Approach

- Visualize / analyze data

Training & Testing

```
id,vendor_id,pickup_datetime,dropoff_datetime,passenger_count,pickup_longitude,pickup_latitude,dropoff_longitude,dropoff_latitude,store_and_fwd_flag,trip_duration
id2875421,2,2016-03-14 17:24:55,2016-03-14 17:32:30,1,-73.982154846191406,40.767936706542969,-73.964630126953125,40.765602111816406,N,455
id2377394,1,2016-06-12 00:43:35,2016-06-12 00:54:38,1,-73.980415344238281,40.738563537597656,-73.999481201171875,40.731151580810547,N,663
id3858529,2,2016-01-19 11:35:24,2016-01-19 12:10:48,1,-73.979026794433594,40.763938903808594,-74.005332946777344,40.710068225097656,N,2124
id3504673,2,2016-04-06 19:32:31,2016-04-06 19:39:40,1,-74.010040283203125,40.719970703125,-74.01226806640625,40.706718444824219,N,429
id2181028,2,2016-03-26 13:30:55,2016-03-26 13:38:10,1,-73.973052978515625,40.793209075927734,-73.972932278808594,40.782520294189453,N,435
id0801584,2,2016-01-30 22:01:40,2016-01-30 22:09:03,6,-73.982856750488281,40.742195129394531,-73.992080688476562,40.749183654785156,N,443
id1813257,1,2016-06-17 22:34:59,2016-06-17 22:40:40,4,-73.969017028808594,40.757839202880859,-73.957405090332031,40.765895843505859,N,341
id1324603,2,2016-05-21 07:54:58,2016-05-21 08:20:49,1,-73.969276428222656,40.797779083251953,-73.922470092773438,40.76055908203125,N,1551
id1301050,1,2016-05-27 23:12:23,2016-05-27 23:16:38,1,-73.999481201171875,40.738399505615234,-73.985786437988281,40.732814788818359,N,255
id0012891,2,2016-03-10 21:45:01,2016-03-10 22:05:26,1,-73.981048583984375,40.744338989257813,-73.972999572753906,40.789989471435547,N,1225
id1436371,2,2016-05-10 22:08:41,2016-05-10 22:29:55,1,-73.982650756835938,40.763839721679688,-74.002227783203125,40.732990264892578,N,1274
id1299289,2,2016-05-15 11:16:11,2016-05-15 11:34:59,4,-73.991531372070313,40.749439239501953,-73.95654296875,40.7706298828125,N,1128
id1187965,2,2016-02-19 09:52:46,2016-02-19 10:11:20,2,-73.962982177734375,40.756679534912109,-73.984405517578125,40.760719299316406,N,1114
id0799785,2,2016-06-01 20:58:29,2016-06-01 21:02:49,1,-73.956306457519531,40.767940521240234,-73.966110229492188,40.763000488828125,N,260
id2900608,2,2016-05-27 00:43:36,2016-05-27 01:07:10,1,-73.992195129394531,40.727226257324219,-73.974655151367188,40.783069610595703,N,1414
id3319787,1,2016-05-16 15:29:02,2016-05-16 15:32:33,1,-73.955513000488281,40.768592834472656,-73.948760986328125,40.771545410156625,N,211
id3379579,2,2016-04-11 17:29:50,2016-04-11 18:08:26,1,-73.991165161132813,40.755561828613281,-73.999290466308594,40.725353240966797,N,2316
id1154431,1,2016-04-14 08:48:26,2016-04-14 09:00:37,1,-73.994255065917969,40.745803833007813,-73.999656677246094,40.723342895507813,N,731
id3552682,1,2016-06-27 09:55:13,2016-06-27 10:17:10,1,-74.003982543945313,40.713012695312500,-73.979194641113281,40.749923706054688,N,1317
id3390316,2,2016-06-05 13:47:23,2016-06-05 13:51:34,1,-73.98388671875,40.738197326660156,-73.991203308105469,40.727870941162109,N,251
id2070428,1,2016-02-28 02:23:02,2016-02-28 02:31:08,1,-73.980369567871094,40.742420196533203,-73.962852478027344,40.760635375976563,N,486
id0809232,2,2016-04-01 12:12:25,2016-04-01 12:23:17,1,-73.979537963867188,40.753360748291016,-73.963996887207031,40.763458251953125,N,652
id2352683,1,2016-04-09 03:34:27,2016-04-09 03:41:30,1,-73.995864868164063,40.758811950683594,-73.993324279785156,40.740322113037109,N,423
id1603037,1,2016-06-25 10:36:26,2016-06-25 10:55:49,1,-73.993553161621094,40.747173309326172,-74.006141662597656,40.704383850097656,N,1163
id3321406,2,2016-06-03 08:15:05,2016-06-03 08:56:30,1,-73.955230712890625,40.777133941650391,-73.788749694824219,40.641471862792969,N,2485
id0129640,2,2016-02-14 13:27:56,2016-02-14 13:49:19,1,-73.956581115722656,40.771358489990234,-73.974967956542969,40.732791900634766,N,1283
```

Training Data




```
id,vendor_id,pickup_datetime,passenger_count,pickup_longitude,pickup_latitude,dropoff_longitude,dropoff_latitude,store_and_fwd_flag
id3004672,1,2016-06-30 23:59:58,1,-73.988128662109375,40.732028961181641,-73.990173339843750,40.756679534912109,N
id3505355,1,2016-06-30 23:59:53,1,-73.964202880859375,40.67999267578125,-73.959808349609375,40.655403137207031,N
id1217141,1,2016-06-30 23:59:47,1,-73.9974365234375,40.737583160400391,-73.986160278320312,40.729522705078125,N
id2150126,2,2016-06-30 23:59:41,1,-73.956069946289063,40.771900177001953,-73.986427307128906,40.73046875,N
id1598245,1,2016-06-30 23:59:33,1,-73.97021484375,40.761474609375,-73.961509704589844,40.755889892578125,N
id0668992,1,2016-06-30 23:59:30,1,-73.991302490234375,40.749797821044922,-73.980514526367188,40.786548614501953,N
id1765014,1,2016-06-30 23:59:15,1,-73.978309631347656,40.741550445556641,-73.952072143554688,40.717002868652344,N
id0898117,1,2016-06-30 23:59:09,2,-74.012710571289063,40.701526641845703,-73.986480712890625,40.719509124755859,N
id3905224,2,2016-06-30 23:58:55,2,-73.992332458496094,40.730510711669922,-73.875617980957031,40.875213623046875,N
id1543102,2,2016-06-30 23:58:46,1,-73.993179321289063,40.748760223388672,-73.979309082031250,40.761310577392578,N
id3024712,1,2016-06-30 23:58:32,4,-73.968528747558594,40.678432464599609,-73.966590881347656,40.635711669921875,N
id3665810,2,2016-06-30 23:58:05,1,-73.982772827148438,40.756908416748047,-73.974693298339844,40.753330230712891,N
id1836461,1,2016-06-30 23:58:01,1,-73.921104431152344,40.767292022705078,-73.936859130859375,40.774044036865234,N
id3457080,2,2016-06-30 23:57:57,1,-73.986801147460938,40.734916687011719,-73.975898742675781,40.756893157958984,N
id3376065,1,2016-06-30 23:57:25,1,-73.996345520019531,40.748161315917969,-73.950828552246094,40.782825469970703,N
id3008739,1,2016-06-30 23:57:22,1,-73.968025207519531,40.762283325195312,-73.934791564941406,40.797435760498047,N
```

Test Data

ML Approach

- The steps involved in the development of a machine learning application involves:
  - Collect data
  - Prepare the input data
  - Analyze the input data
  - Train the algorithm
  - Test the algorithm
  - Predict

Datasets

https://www.data.gov/

- US-centric agriculture, climate, education, energy, finance, health, manufacturing data, …

https://cloud.google.com/bigquery/public-data/

- BigQuery (Google Cloud) public datasets (bikeshare, GitHub, Hacker News, Form 990 non-profits, NOAA, …)

https://www.kaggle.com/datasets

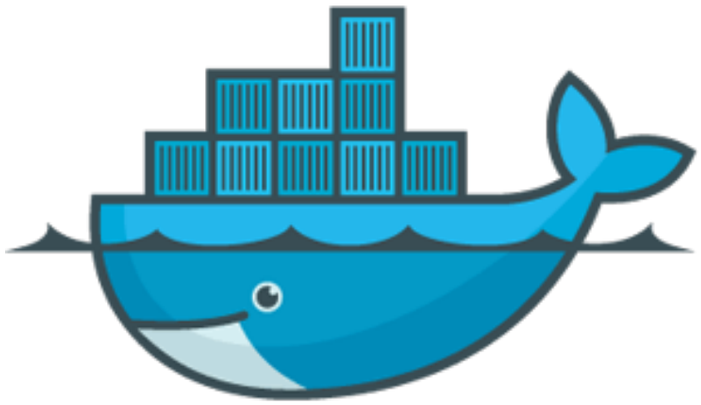
- Microsoft-owned, various (Billboard Top 100 lyrics, credit card fraud, crime in Chicago, global terrorism, world happiness, …)

https://aws.amazon.com/public-datasets/

- AWS-hosted, various (NASA, a bunch of genome stuff, Google Books n-grams, Multimedia Commons, …)

Some Technologies we will use

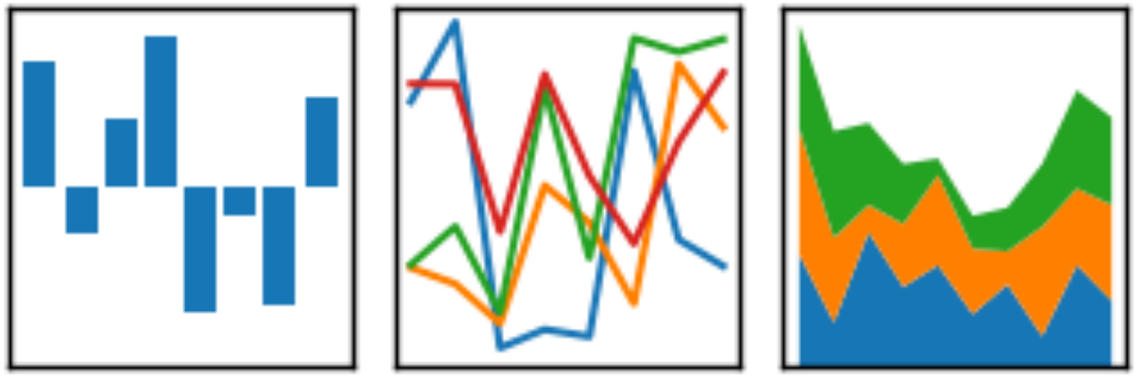
python™



docker

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



Data - Example

Outlook	Temperature	Humidity	Windy	Play
Sunny	hot	high	false	no
Sunny	hot	high	true	no
Overcast	hot	high	false	yes
Rainy	mild	high	false	yes
Rainy	cool	normal	false	yes
Rainy	cool	normal	true	no
Overcast	cool	normal	true	yes
Sunny	mild	high	false	no
Sunny	cool	normal	false	yes
Rainy	mild	normal	false	yes
Sunny	mild	normal	true	yes
Overcast	mild	high	true	yes
Overcast	hot	normal	false	yes
Rainy	mild	high	true	no

Sparse Data

- Most of the attributes have a value of 0 for most of the instances.

- For example, items purchased in a store is zero for most of the items.

- The data matrix with rows as customers and columns as items, is sparse.

Sparse Data

Items

ID	Item
0	Bread
1	Milk
2	chips
3	salsa
4	Eggs
5	Coke

Transactions

TID	Items
1	Bread, milk
2	chips
3	Milk, Eggs
4	chips, salsa
5	Bread,Eggs,Coke

Data:

T1: {a, b, 0,0,0,0}

T2: {0,0,c,0,0,0}

T3: {0,d,0,0,e,0}

T4: {0,0,f,g,0,0}

T5: ….

{(0,a),(1,b)}

{(2,c)}

{(1,d),(4,e)}

{(2,f),(3,g)}

{(0,h),(4,i),(5,j)}

Missing Values

- Missing values are frequently indicated by out-of-range entries, for example,

  - negative numbers that is normally only positive

  - a 0 in a numeric field that can never be normally 0

  - missing values may also be indicated by blanks and dashes.

Normalization

- Attributes are often normalized to lie in a fixed range - usually from 0 to 1, for example

- dividing all the values by the maximum value encountered,

  or

- by subtracting the minimum value and dividing by the range between the maximum and minimum values.

Normalization

- Another normalization technique is to:

  - calculate the statistical mean and the standard deviation of the attribute values,

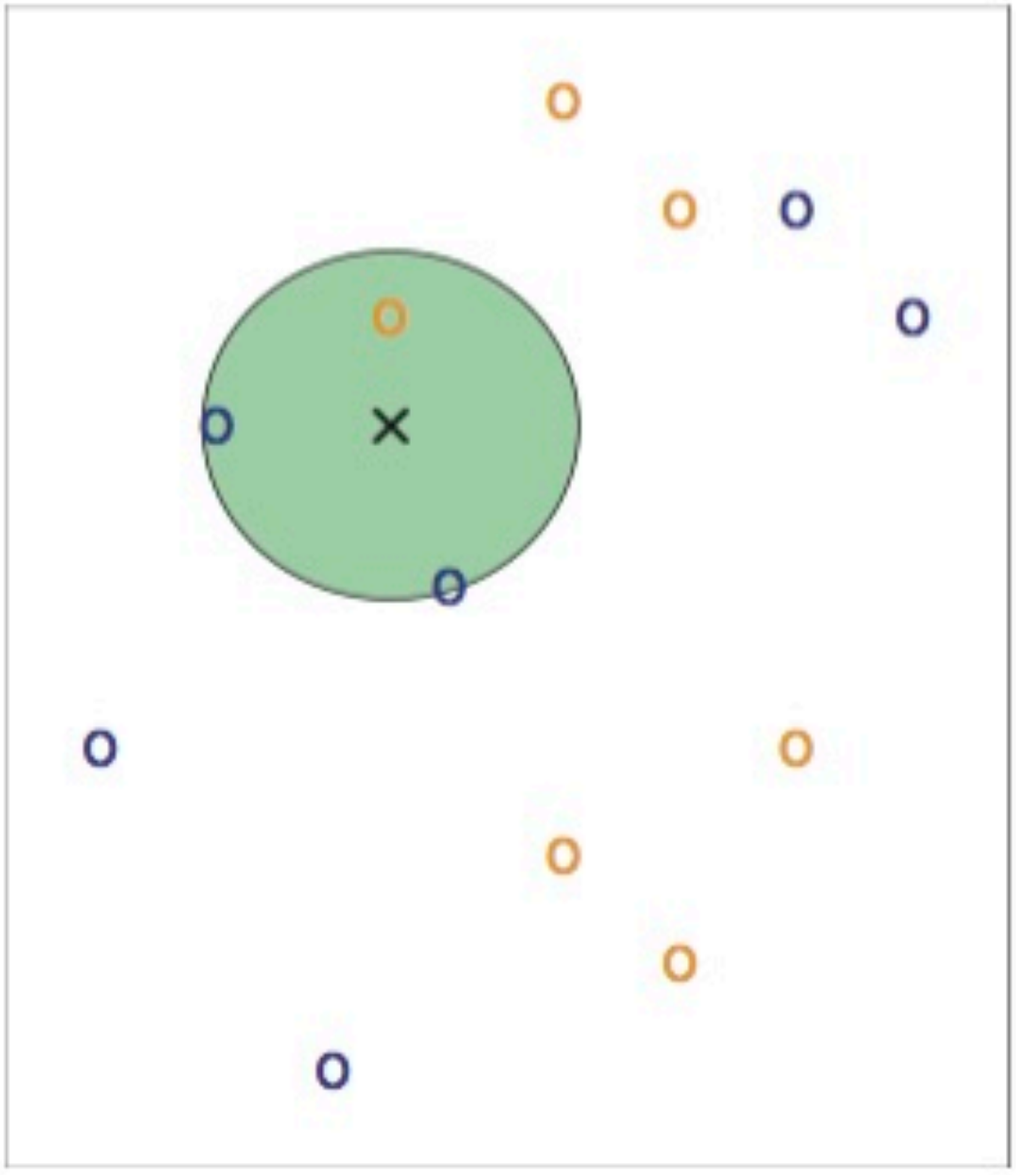
  - then subtract the mean from each value, and,

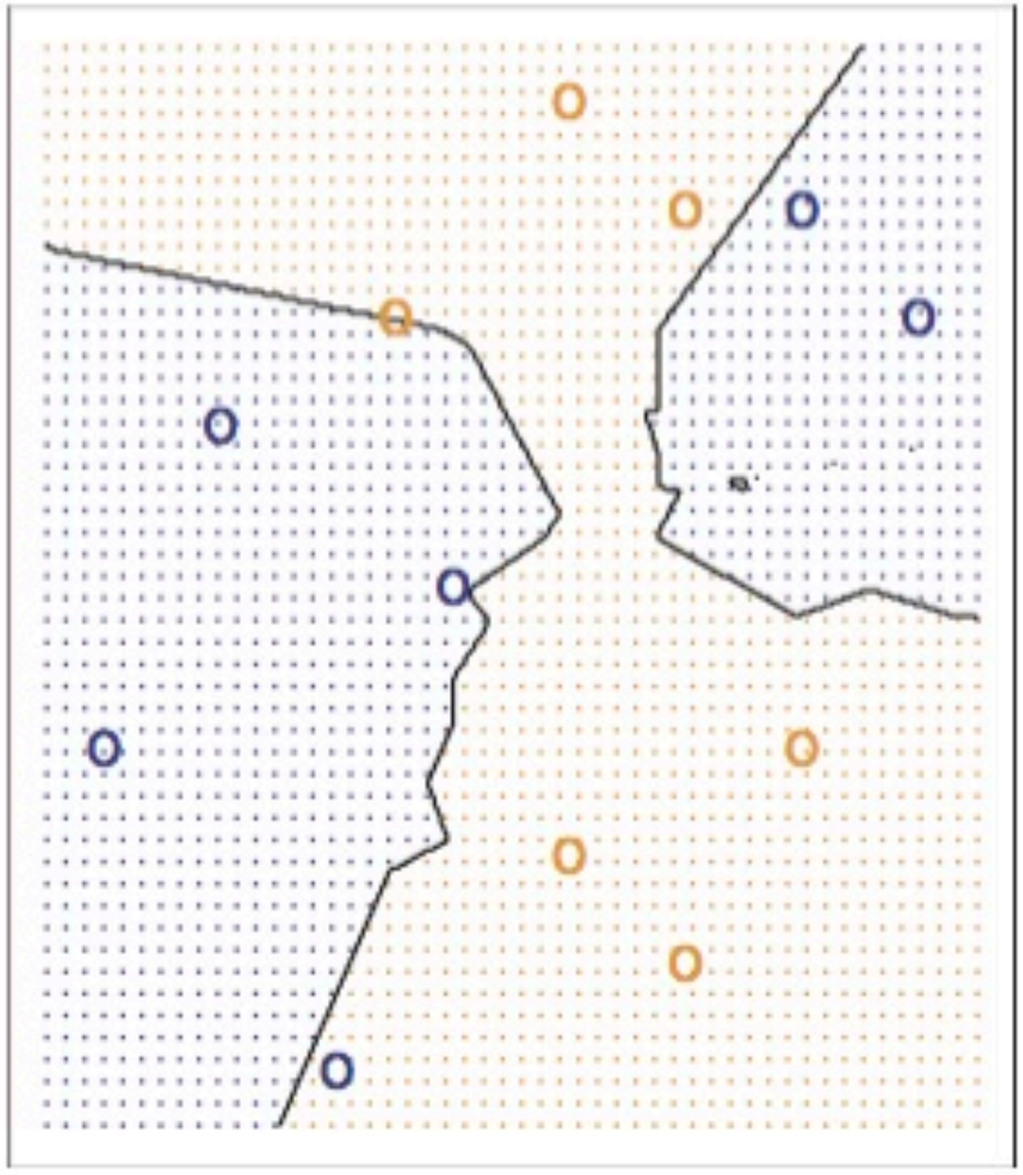
  - divide the result by the standard deviation.

- This process is called standardizing a statistical variable.

- It results into a set of values whose mean is 0 and the standard deviation is 1.

K-Nearest Neighbor Classification





BUT FIRST, SNAKES!

Python is an interpreted, dynamically-typed, high-level, garbage-collected, object-oriented-functional-imperative, and widely used scripting language.

- **Interpreted:** instructions executed without being compiled into (virtual) machine instructions*
- **Dynamically-typed:** verifies type safety at runtime
- **High-level:** abstracted away from the raw metal and kernel
- **Garbage-collected:** memory management is automated
- **OOFI:** you can do bits of OO, F, and I programming

Not the point of this class!

- Python is fast (developer time), intuitive, and used in industry!

*you can compile Python source, but it's not required

THE ZEN OF PYTHON

- 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.

Thanks: SDSMT ACM/LUG

LITERATE PROGRAMMING

Literate code contains in one document:

- the source code;
- text explanation of the code; and
- the end result of running the code.

Basic idea: present code in the order that logic and flow of human thoughts demand, not the machine-needed ordering

- Necessary for data science!
- Many choices made need textual explanation, ditto results.

Stuff you'll be using in Project 1 (and beyond)!

IP[y]: IPython Interactive Computing

jupyter

Interpreter

- Reads a program line and executes it

Compiler

- Reads a program completely, translates it into executable code and then executes

Program

- A sequence of instructions in a particular language.
- The details will be different in different languages.
- In every program, there is an input, an output
- In between there are some mathematical operations, or conditional execution and also repetition.

Debugging

- There will always be bugs.
- Process of tracking down the bugs is called debugging.
- There are three types of errors:
  - Syntax errors
  - Runtime errors, also called exceptions
  - Semantic errors - the meaning of the program is wrong (for example using % instead of / for floating point division)

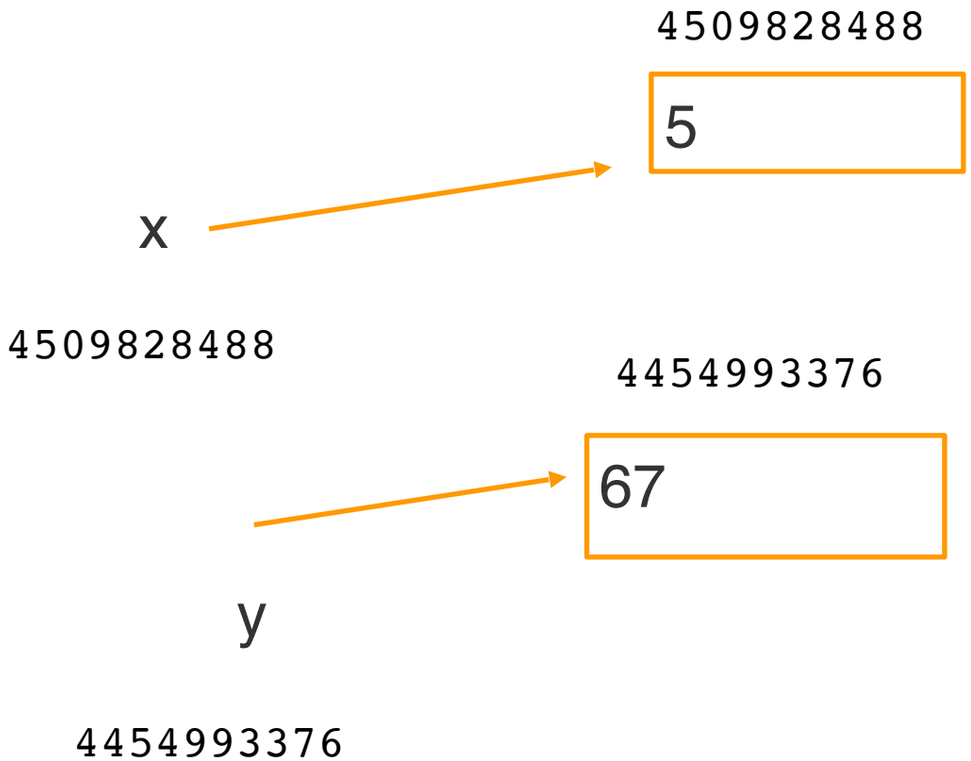
Variables, Expressions and Statements

Values

- A basic unit of a program
- Types:
  - integers
  - strings
  - float
  - complex

Variables

- A powerful feature of a programming language.
- A variable refers to a value.
- An assignment operator creates a new variable and assigns a value


4509828488
5
x
4509828488
4454993376
67
y
4454993376

Variable Names and Keywords

- Variable names can be both letters and numbers.
- They have to begin with a letter though.
- Uppercase letters are allowed, generally we use lowercase variable names.
- Underscore character, _ , can also appear in a name.
- An illegal name results in a syntax error.

PYTHON 2 VS 3

Python 3 is intentionally backwards incompatible

- (But not *that* incompatible)

Biggest changes that matter for us:

- `print "statement"` → `print("function")`
- `1/2 = 0` → `1/2 = 0.5` and `1//2 = 0`
- ASCII `str` default → default Unicode

Namespace ambiguity fixed:

```
i = 1
[i for i in range(5)]
print(i)   # ????????
```

TO ANY CURMUDGEONS …

If you're going to use Python 2 anyway, use the `_future_` module:

- Python 3 introduces features that will throw runtime errors in Python 2 (e.g., `with` statements)
- `_future_` module incrementally brings 3 functionality into 2
- https://docs.python.org/2/library/__future__.html

```
from _future_ import division
from _future_ import print_function
from _future_ import please_just_use_python_3
```

Operators and operands

- Operators represent computations like addition, multiplication, division etc.

  *, -, +, / and **

- The values the operator is applied to are called operands.

Expressions and Statements

- An expression is a combination of values, variables, and operators.
- A value all by itself is also an expression
- Variable is also an expression
  for e.g.,
  21
  x
  x + 21
- A statement is a unit of code that a Python interpreter can execute. e.g., print, and assignment

Order of Operations

- When more than one operator appears in an expression, the order of evaluation depends on the rules of precedence.
- The acronym of precedence is PEMDAS
  - Parentheses (1+1)**(5-3)
  - Exponentiation, 2**1+1
  - Multiplication and Division have the same precedence,
  - Addition and subtraction have the same precedence
  - Operators with the same precedence are evaluated from left to right (except exponentiation) for e.g., 2/5*3.

String Operations

- You can't perform mathematical operations on strings.
  '2' - '1' 'eggs'/'dozens'
- The '+' operator works with strings and performs concatenation.
  first = "hello"
  second = "Class"
  print(first + second)
- The output is "helloclass"

String Operations

- The ‘*’ operator works with strings and performs repetition.
  ‘Spam’*3 is ‘SpamSpamSpam’
- If one of the operands is a string, the other has to be an integer.

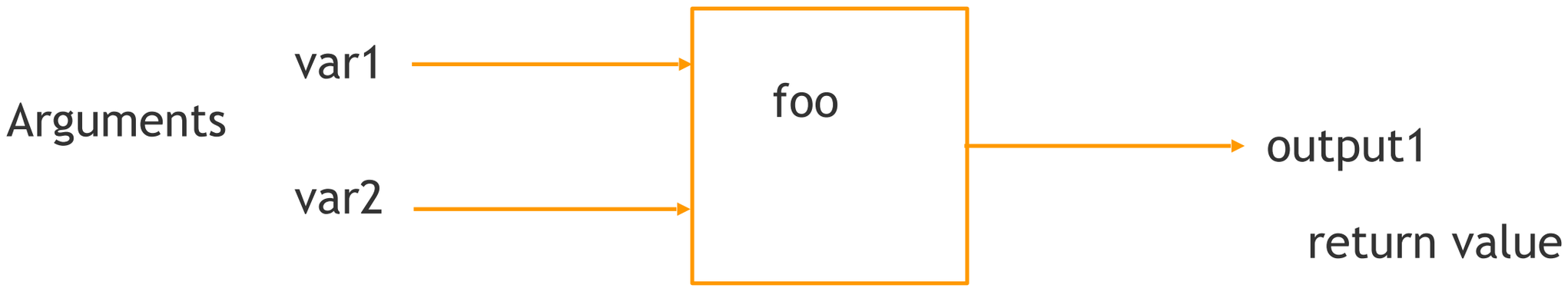
Comments

- As programs get bigger and more complicated, they get more difficult to read.
- It is a good idea to add notes to your programs.
- The comments start with # symbol

Functions

Function as a Black Box

```
def foo(a,b):
    c = a+b
    return c
```



encapsulation- information hiding

m= foo(d,e)

Functions

- A named sequence of statements that perform a certain computation.
- Later on, you can call the function by name
- for example, type(34)

Why Functions?

- A group of statements gets a name
- Modular code
- Easier Debugging
- Code Reuse

Type Conversion Functions

- There are built-in functions in Python that convert from one type to another.
- The int function takes any value and converts it to an integer.
- int function takes any value and converts it to an integer, if it can.
- int can convert floating-point values to integers, but it doesn't round off. It chops off the fraction part.

Type Conversion

- float converts integers and strings to floating-point numbers
- str converts its arguments to string

Math Functions

- Python has a math module that provides mathematical functions.
- Before we can use the module, we have to import it:
  import math
- This module contains the functions and variables defined in the module.
- To access one of the functions, you have to specify the name of the module and the name of the function.

USEFUL BUILT-IN FUNCTIONS: COUNTING AND ITERATING

`len`: returns the number of items of an enumerable object

```
len( [‘c’, ‘m’, ‘s’, ‘c’, 3, 2, 0] )
```

```
7
```

`range`: returns an iterable object

```
list( range(10) )
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

`enumerate`: returns iterable tuple (index, element) of a list

```
enumerate( [“311”, “320”, “330”] )
```

```
[(0, “311”), (1, “320”), (2, “330”)]
```

https://docs.python.org/3/library/functions.html

USEFUL BUILT-IN FUNCTIONS: MAP AND FILTER

`map`: apply a function to a sequence or iterable

```
arr = [1, 2, 3, 4, 5]
map(lambda x: x**2, arr)
```

```
[1, 4, 9, 16, 25]
```

`filter`: returns a list of elements for which a predicate is true

```
arr = [1, 2, 3, 4, 5, 6, 7]
filter(lambda x: x % 2 == 0, arr)
```

```
[2, 4, 6]
```

We'll go over in much greater depth with pandas/numpy.

User defined Functions

- Use keyword def

```
def message():
        print(“Hello Class”)
```

Flow of Execution

- The program flow
- Follow the flow of execution.

Parameters and Arguments

- Some built-in functions require arguments.
- for example , math functions
- Inside the function, arguments are assigned to variables called parameters.
- An expression can also be used as an argument.

Variables and parameters

- A variable created inside a function is local.
- Parameters are also local
- Some functions return a value and others are void which return nothing.

Import

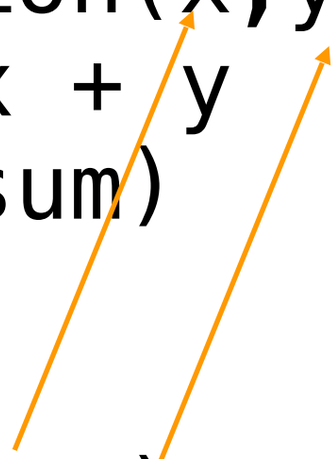
- Python provides two ways to import modules
- We have already seen import math
- We can also import functions using from

Code

```
def summation(x,y):
    sum = x + y
    print(sum)



summation(2,4)
```



x = 2
y = 4

```
def summation(x,y):
    z = x + y
    return(z)


print(summation(2,4))
```

```
def summation(x,y):
    z = x + y
    return(z)


m = summation(2,4)
print(m)
```

Conditionals

Import

- Python provides two ways to import modules
- We have already seen import math
- We can also import functions using from

Modulus Operator

- Modulus operator works on integers.
- It yields a remainder when the first operator is divided by the second.
- Modulus operator is a percent sign (%)
- quotient = 7 // 3
- remainder = 7%3

Modulus Operator

- Usefulness of modulus operator?

Modulus Operator

- Usefulness of modulus operator?

  If I ask you to find the last digit of a number or more.

```
97856 % 100
```

```
56
```

Boolean Expressions

- A boolean expression is an expression that is either true or false.
- It uses the operator '==', which compares the operands to the left and the right of this operator
- It produces either True or False

  for example, 5==5 will return True
  and 5==6 will return False
- True and False are of type bool

Other relational Operators

- x != y # x is not equal to y
- x > y # x is greater than y
- x < y # x is less than y
- x >= y # x is greater than or equal to y
- x <= y # x is less than or equal to y

x = y

Logical Operators

- Three logical operators: **and**, **or**, and **not**
  for example,
  - x > 0 and x < 10
    This is true only if x is greater than 0 and less than 10
  - n%2 == 0 or n%3 == 0 is true if either of the conditions is true, that is, if the number is divisible by 2 or 3.
  - The not operator negates a boolean expression, so not(x>y) is true if x > y is false, that is, if x is less than or equal to y.

Conditional Execution

- We need to check conditions and change the behavior of the program

- The simplest conditional statement is if, for example

```
if x>0:
    print('x is positive')
```

```
x is positive
```

- The boolean expression after ‘if’ is called the condition.

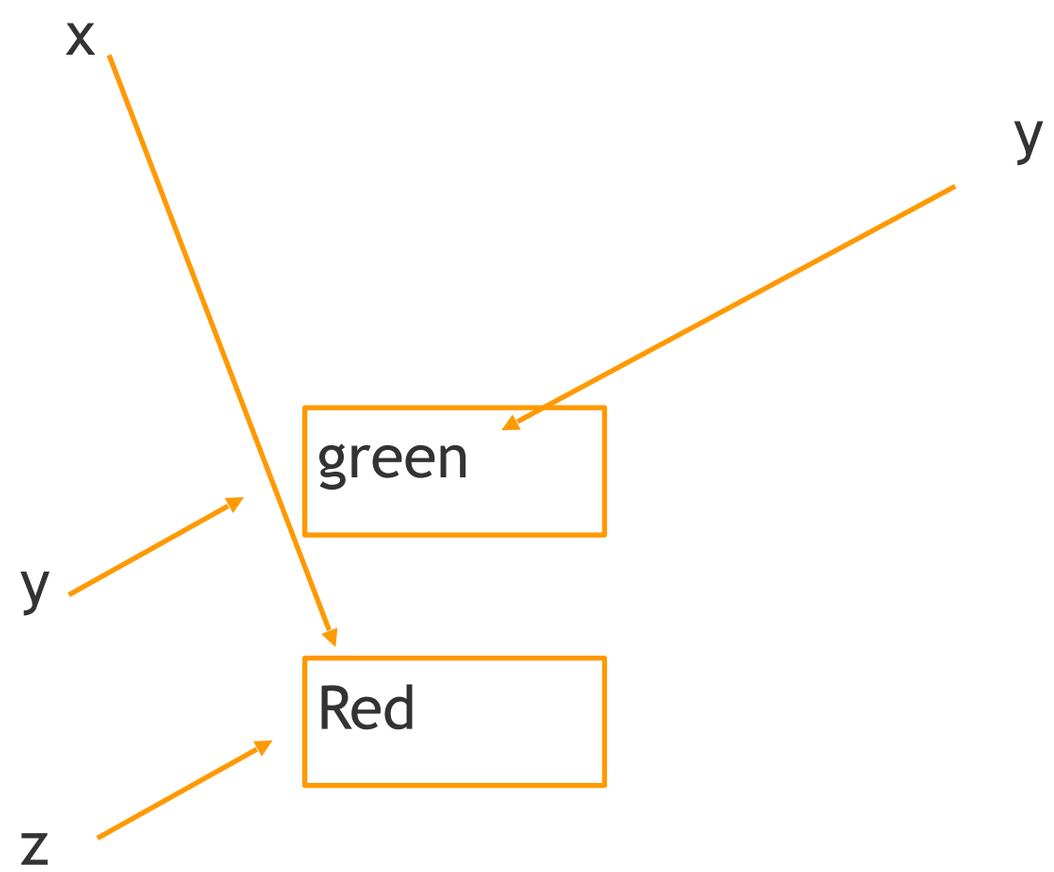
```
def decide(x,y):
    if x=='green' and y=='Red':
        print("You can go!")



y = 'Red'
y = 'green'
y = 'green'

z = 'Red'

decide(z,y)
```



if condition

- If statements have the same structure as function definitions: a header followed by an indented body.

- Statements like these are called compound statements.

- There is no limit on the number of statements that can appear in the body, but there has to be at least one.

if condition

- Sometimes it is useful to have a body with no statements, usually as a place holder

- In that case use the pass statement, which does nothing

```
if x < 0:
    pass        # need to handle negative values
```

Alternative execution

- When there are two possibilities and the condition determines which one gets executed

```
if x%2 == 0:
    print('x is even')
else:
    print('x is odd')
```

- The alternatives are called branches, because they are branches in the flow of execution.

Chained Conditionals

- Sometimes there are more than two possibilities and we need more than two branches.
- One way to express a computation like that is a chained conditional:

```
if(x < y):
    print('x is less than y')
elif (x > y):
    print('x is greater than y')
else:
    print('x and y are equal')
```

elif is an abbreviation of “else if”

Chained Conditionals

- There is no limit to the number of elif statements.
- If there is an else clause, it has to be at the end.
- We don’t need an else statement

```
if choice == 'a':
    draw_a()
elif choice == 'b':
    draw_b()
elif choice == 'c':
    draw_c()
```

- Each condition is checked in order. If the first one is false, the next is checked, and so on.

Iterations

Iterations

- It involves repetition.
- A statement or a group of statements that need to be repeated.
- Help in automation of repetitive tasks.

for statement

- A ‘for’ statement requires an iterator and starts with the keyword ‘for’

range from 0 to 10

```
for i in range(10):
    print(i)
```

- This iteration will start with a 0 and the last number is not included

for statement with a range

- ‘for’ statement can be used to display a specific range of numbers

range from 40 to 49

```
for i in range(40,50):
    print(i)
```

- This iteration will start with 40 and the last number is not included

for statement with a range and steps

- ‘for’ statement can be used to display a specific range of numbers in different steps

numbers from 40 to 49 in steps of 2

```
for i in range(40,50,2):
    print(i)
```

- This iteration will start with 40 and the last number is not included

while statement

- It requires a condition, for e.g;

```
n=12
while(n>0):
    print(n)
```

- In the definition there is no starting iterator.

- We need to use a condition inside while

while statement with a condition

- It requires a condition, for e.g;

```
n=12
while(n>0):
    print(n)
    n = n - 1
```

- At the start of each iteration the condition is checked.

flow of execution for a while statement

- Evaluate the condition, yielding True or False.
- If the condition is false, exit the while statement and continue execution at the next statement
- if the condition is true, execute the body and then go to the condition again.
- The body of the loop will change the value of one or more variables so that condition eventually becomes false and the loop terminates.

break statement

- Sometimes we want to break out of a loop if some value is seen

```
n=12
while(n>0):
    print(n)
    n = n - 1
    if n == 7:
        break
```

```
12
11
10
9
8
```

Strings

String

- A sequence of characters.

  course = “MSML 605”
- You can access the characters one at a time with the bracket operator:

  second_character = course[1]

```
course = "MSML 605"
second_course = course[1]
print(second_course)
```

```
S
```

- index has to be an integer.

strings

- len returns the number of characters in a string

```
course = "MSML 605"
second_course = course[1]
print(second_course)
```

```
s
```

```
len(course)
```

```
8
```

- last index using length

strings

- Using negative indices

course[-1]

```
course = "MSML 605"
second_course = course[1]
print(second_course)
```

```
S
```

```
course[-1]
```

```
'5'
```

Traversal

- Processing one character at a time:

```
course = 'MSML 605'
index = 0
while index < len(course):
    letter = course[index]
    print(letter)
    index += 1
```

```
M
S
M
L

6
0
5
```

- Displays each character on a separate line

Traversal

- A more Pythonic way to traverse a string using for:

```
course = 'MSML 605'
for c in course:
    print(c)
```

```
M
S
M
L

6
0
5
```

```
course = 'MSML 605'
for c in course:
    print(c,end=' ')
```

```
M S M L   6 0 5
```

String Operations

String Concatenation

- Two strings can be concatenated using a ‘+’ operator

```
word1 = 'abc'
word2 = 'def'
word = word1 + word2
print(word)
```

```
abcdef
```

String Slices

- A segment of a string is a slice
- Selecting a slice is similar to selecting a character

```
s = "Monty Python"
print(s[0:5])
print(s[6:12])
```

```
Monty
Python
```

- The operator [n:m] returns the part of the string from the “n-eth” character to the “m-eth” character.
- It includes the first but excludes the last.

String Slices

- If the first index before the colon is omitted, the slice starts at the beginning of the string.
- If you omit the second index, the slice goes to the end of the string

```
course = "MSML 605"
print(course[:3])
print(course[2:])
```

```
MSM
ML 605
```

- If the first index >= second index, the result is an empty string.

Strings are immutable

- What happens if [ ] operators are used on the left side of the assignment operator?

```
greeting = 'hello world'
greeting[0] = 'J'
```

- You can create a new string
  new_greeting = 'J' + greeting[1:]

```
new_greeting = 'J' + greeting[1:]
new_greeting
```

```
'Jello world'
```

- It does not change the original string

String Search

- What does the following function do?

```
def find(word,letter):
    index = 0
    while index < len(word):
        if word[index] == letter:
            return(index)
        index += 1
    return(-1)
```

String Search

- What does the following function do?

```
def find(word,letter):
    index = 0
    while index < len(word):
        if word[index] == letter:
            return(index)
        index += 1
    return(-1)
```

- find is the opposite of the [ ] operator.
- Instead of taking an index and extracting the corresponding character, it takes a character and finds an index

Looping and Counting

- The following program counts the number of times the letter ‘a’ appears in a string:

```
word = 'banana'
count = 0
for letter in word:
    if letter == 'a':
        count += 1
print(count)
```

```
3
```

String Methods

- A method is similar to a function - it takes arguments and returns a value.
- The syntax is different, for example

```
word = 'banana'
new_word = word.upper()
print(new_word)
```

```
BANANA
```

- A method call is called an invocation
- We are invoking upper on the word.

String Methods

- There is a string method named ‘find’

```
word = ‘banana’
index = word.find(‘a’)
print(index)
```

- We invoke find on word.
- find can also find substrings not just characters

```
word.find(‘na’)
```

String Methods

- It can take as a second argument the index where it should start:

  word.find(‘na’,3)

- As a third argument the index where it should stop:

  name = ‘bob’
  name.find(‘b’,1,2)

in Operator

- The word ‘in’ is a boolean operator that takes two strings and returns True if the first appears as a substring in the second

  ‘a’ in ‘banana’

  ‘seed’ in ‘banana’

String Comparison

- Relational operators work on strings
- Equality operator ‘==’
- Other relational operations are useful for putting words in alphabetical order:

```
if word < 'banana':
    print('Your word, '+word+', comes before banana.')
elif word > 'banana':
    print('Your word,' + word + ', comes after banana.')
else:
    print('All right, bananas.’)
```

- Uppercase letters come before all the lowercase letters.

Lists

List

- A list is a sequence
- A sequence of values
- These values can be of any type.
- Values in a list are called items or elements.

Lists

- The simplest way to create a list is to enclose elements in square brackets ([and])

  [10, 12, 14, 15]
  [‘Tom cat’, ‘Jerry mouse’]
  [‘spam’, 21, ‘a’, 34.5]

- You can assign list values to variables
  data = [10, 12, 14, 15]

- Even an empty list, arr = [ ]

Lists are mutable

- The syntax for accessing elements is the same as for accessing string characters.
- The expression inside brackets specifies the index.

```
numbers = [7, 34, 56]
numbers[1] = 36
print(numbers)
```

Mapping

- You can think of a list as a relationship between indices and elements.
- This relationship is called mapping
- Each index “maps to” one of the elements.

num = [2, 34, 56]

List Indices

- List indices work the same way as string indices:
  - Any integer expression can be used as an index
  - if you try to eat or write an element that does not exist, you get an IndexError
  - If an index has a negative value, it counts backward from the end of the list.

‘in’ operator

- The ‘in’ operator also works on lists.

  cheeses = [‘Cheddar’, ‘Mozzarella’, ‘Blue’]
  ‘Blue’ in cheeses
  ‘Brie’ in cheeses

Traversing a list

- The most common way is with a ‘for’ loop
- Syntax is the same as for strings

```
cheeses = [‘Cheddar’, ‘Mozzarella’, ‘Blue’]
for cheese in cheeses:
    print(cheese)
```

- This works well if you only need to read the elements.

Traversing a list

- If you want to write or update the elements, you need the indices.
- Common way is to combine functions ‘range’ and ‘len'

```
for i in range(len(numbers)):
        numbers[i] = numbers[i] * 2
```

- This loop traverses the list and updates each element.

Nested lists

- A list can contain another list

  [‘spam’, 1, [‘Brie’, 5, 3.2],2,[2,5,6]]

- Each internal list still counts as a single element.

List Operations

- ‘+’ operator concatenates lists

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = a + b
print(c)
```

```
[1, 2, 3, 4, 5, 6]
```

- ‘*’ operator repeats a list given number of times

```
>>> [0] * 4
[0, 0, 0, 0]
>>> [1, 2, 3] * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

List Slices

- Slice operator also works on lists:

```
>>> t = [‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’]
>>> t[1:3]
[‘b’, ‘c’]
>>> t[:4]
[‘a’, ‘b’, ‘c’, ‘d’]
>>> t[3:]
[‘d’, ‘e’, ‘f’]
```

- If you omit the first index, the slice starts at the beginning
- If you omit the second, the slice goes to the end.
- If you omit both, the slice is a copy of the whole list

```
>>> t[:]
[‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’]
```

List Methods

- Python provides methods that operate on lists
- append adds a new element to the end of a list

```
>>> t = [‘a’, ‘b’, ‘c’]
>>> t.append(‘d’)
>>> print(t)
[‘a’, ‘b’, ‘c’, ‘d’]
```

List Methods

- `extend` takes a list as an argument and appends all of the elements:

```
>>> t1 = ['a', 'b', 'c']
>>> t2 = ['d', 'e']
>>> t1.extend(t2)
>>> print(t1)
['a', 'b', 'c', 'd', 'e']
```

- t2 is unmodified

List Methods

- `sort` arranges the elements of the list from low to high:

```
>>> t = ['d', 'b', 'c', 'a', 'e']
>>> t.sort()
>>> print(t)
['a', 'b', 'c', 'd', 'e']
```

- List methods are all void; they modify the list and return None

List Methods

- `sort` arranges the elements of the list from low to high:

```
>>> t = ['d', 'b', 'c', 'a', 'e']
>>> t.sort()
>>> print(t)
['a', 'b', 'c', 'd', 'e']
```

- List methods are all void; they modify the list and return None