Netflix intro

What data should we encode about each Netflix account holder to help us make effective recommendations?

In machine learning, clustering can be used to group similar data for prediction and recommendation. For example, each Netflix user's viewing history can be represented as a n-tuple indicating their preferences about movies in the database, where n is the number of movies in the database. People with similar tastes in movies can then be clustered to provide recommendations of movies for one another. Mathematically, clustering is based on a notion of distance between pairs of n-tuples.

Data types

Term	**Examples**: (add additional examples from class)
set unordered collection of elements *repetition doesn't matter* *Equal sets agree on membership of all elements*	$7 \in \{43, 7, 9\}$ $\quad 2 \notin \{43, 7, 9\}$
n-tuple ordered sequence of elements with n "slots" $(n > 0)$ *repetition matters, fixed length* *Equal n-tuples have corresponding components equal*	
string ordered finite sequence of elements each from specified set *repetition matters, arbitrary finite length* *Equal strings have same length and corresponding characters equal*	

Special cases:

When $n = 2$, the 2-tuple is called an **ordered pair**.

A string of length 0 is called the **empty string** and is denoted λ.

A set with no elements is called the **empty set** and is denoted $\{\}$ or $\emptyset$.

Set operations

To define a set we can use the roster method, set builder notation, a recursive definition, and also we can apply a set operation to other sets.

New! Cartesian product of sets and **set-wise concatenation of sets of strings**

Definition: Let X and Y be sets. The **Cartesian product** of X and Y, denoted $X \times Y$, is the set of all ordered pairs (x, y) where $x \in X$ and $y \in Y$

$$X \times Y = \{(x, y) \mid x \in X \text{ and } y \in Y\}$$

Definition: Let X and Y be sets of strings over the same alphabet. The **set-wise concatenation** of X and Y, denoted $X \circ Y$, is the set of all results of string concatenation xy where $x \in X$ and $y \in Y$

$$X \circ Y = \{xy \mid x \in X \text{ and } y \in Y\}$$

Pro-tip: the meaning of writing one element next to another like xy depends on the data-types of x and y. When x and y are strings, the convention is that xy is the result of string concatenation. When x and y are numbers, the convention is that xy is the result of multiplication. This is (one of the many reasons) why is it very important to declare the data-type of variables before we use them.

Fill in the missing entries in the table:

Set	**Example elements in this set**:			
B	`A`	`C`	`G`	`U`
	$(\texttt{A}, \texttt{C})$		$(\texttt{U}, \texttt{U})$	
$B \times \{-1, 0, 1\}$				
$\{-1, 0, 1\} \times B$				
			$(0, 0, 0)$	
$\{\texttt{A}, \texttt{C}, \texttt{G}, \texttt{U}\} \circ \{\texttt{A}, \texttt{C}, \texttt{G}, \texttt{U}\}$				
			`GGGG`	

Defining functions

New! Defining functions A function is defined by its (1) domain, (2) codomain, and (3) rule assigning each element in the domain exactly one element in the codomain.

The domain and codomain are nonempty sets.
The rule can be depicted as a table, formula, or English description.
The notation is

"Let the function FUNCTION-NAME: DOMAIN $\to$ CODOMAIN be given by
FUNCTION-NAME(x) = ... for every $x \in DOMAIN$".

or

"Consider the function FUNCTION-NAME: DOMAIN $\to$ CODOMAIN given by
FUNCTION-NAME(x) = ... for every $x \in DOMAIN$".

Example: The absolute value function

Domain

Codomain

Rule

Defining functions recursively

When the domain of a function is a *recursively defined set*, the rule assigning images to domain elements (outputs) can also be defined recursively.

Recall: The set of RNA strands S is defined (recursively) by:

Basis Step:	$\mathtt{A} \in S, \mathtt{C} \in S, \mathtt{U} \in S, \mathtt{G} \in S$
Recursive Step:	If $s \in S$ and $b \in B$, then $sb \in S$

where sb is string concatenation.

Definition (Of a function, recursively) A function *rnalen* that computes the length of RNA strands in S is defined by:

		$rnalen : S$	$\to \mathbb{Z}^+$
Basis Step:	If $b \in B$ then	$rnalen(b)$	$= 1$
Recursive Step:	If $s \in S$ and $b \in B$, then	$rnalen(sb)$	$= 1 + rnalen(s)$

The domain of *rnalen* is

The codomain of *rnalen* is

Example function application:

$$rnalen(\mathtt{ACU}) =$$

Extra example: A function *basecount* that computes the number of a given base b appearing in a RNA strand s is defined recursively: *fill in codomain and sample function applications*

		$basecount : S \times B$	$\to$	
Basis Step:	If $b_1 \in B, b_2 \in B$	$basecount(b_1, b_2)$	$= \begin{cases} 1 & \text{when } b_1 = b_2 \\ 0 & \text{when } b_1 \neq b_2 \end{cases}$	
Recursive Step:	If $s \in S, b_1 \in B, b_2 \in B$	$basecount(sb_1, b_2)$	$= \begin{cases} 1 + basecount(s, b_2) & \text{when } b_1 = b_2 \\ basecount(s, b_2) & \text{when } b_1 \neq b_2 \end{cases}$	

$basecount(\mathtt{ACU}, \mathtt{A}) =$

$basecount(\mathtt{ACU}, \mathtt{G}) =$

Extra example: The function which outputs 2^n when given a nonnegative integer n can be defined recursively, because its domain is the set of nonnegative integers.

Why represent numbers

Modeling uses data-types that are encoded in a computer.

The details of the encoding impact the efficiency of algorithms we use to understand the systems we are modeling and the impacts of these algorithms on the people using the systems.

Case study: how to encode numbers?

Base expansion definition

Definition For b an integer greater than 1 and n a positive integer, the **base b expansion of** n is

$$(a_{k-1} \cdots a_1 a_0)_b$$

where k is a positive integer, $a_0, a_1, \ldots, a_{k-1}$ are nonnegative integers less than b, $a_{k-1} \neq 0$, and

$$n = a_{k-1} b^{k-1} + \cdots + a_1 b + a_0$$

Notice: *The base b expansion of a positive integer n is a string over the alphabet $\{x \in \mathbb{N} \mid x < b\}$ whose leftmost character is nonzero.*

Base b	Collection of possible coefficients in base b expansion of a positive integer
Binary ($b = 2$)	$\{0, 1\}$
Ternary ($b = 3$)	$\{0, 1, 2\}$
Octal ($b = 8$)	$\{0, 1, 2, 3, 4, 5, 6, 7\}$
Decimal ($b = 10$)	$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
Hexadecimal ($b = 16$)	$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$ letter coefficient symbols represent numerical values $(A)_{16} = (10)_{10}$ $(B)_{16} = (11)_{10}$ $(C)_{16} = (12)_{10}$ $(D)_{16} = (13)_{10}$ $(E)_{16} = (14)_{10}$ $(F)_{16} = (15)_{10}$

Base expansion examples

Common bases:	Binary $b = 2$	Octal $b = 8$	Decimal $b = 10$	Hexadecimal $b = 16$

Examples:

$(1401)_2$

$(1401)_{10}$

$(1401)_{16}$

Algorithm definition

New! An algorithm is a finite sequence of precise instructions for solving a problem.

Algorithm half

Algorithm for calculating integer part of half the input

```
1  procedure half(n: a positive integer)
2  r := 0
3  while n > 1
4     r := r + 1
5     n := n − 2
6  return r {r holds the result of the operation}
```

n	r	$n > 1$?
6		

n	r	$n > 1$?
5		

Algorithm log

Algorithm for calculating integer part of log

```
1  procedure log(n: a positive integer)
2  r := 0
3  while n > 1
4     r := r + 1
5     n := half(n)
6  return r {r holds the result of the log operation}
```

n	r	$n > 1$?
8		

n	r	$n > 1$?
6		

$2^0 = 1$ $2^1 = 2$ $2^2 = 4$ $2^3 = 8$ $2^4 = 16$ $2^5 = 32$ $2^6 = 64$ $2^7 = 128$ $2^8 = 256$ $2^9 = 512$ $2^{10} = 1024$

Division algorithm

Integer division and remainders (aka The Division Algorithm) Let n be an integer and d a positive integer. There are unique integers q and r, with $0 \leq r < d$, such that $n = dq + r$. In this case, d is called the divisor, n is called the dividend, q is called the quotient, and r is called the remainder. We write $q = n$ **div** d and $r = n$ **mod** d.

Extra example: How do **div** and **mod** compare to / and % in Java and python?

Base expansion algorithms

Two algorithms for constructing base b expansion from decimal representation

Most significant first: Start with left-most coefficient of expansion

Calculating integer part of $\log_b$

```
1  procedure logb(n,b: positive integers with b>1)
2  r := 0
3  while n > 1
4     r := r+1
5     n := n div b
6  return r {r holds the result of the log_b operation}
```

Calculating base b expansion, from left

```
1  procedure baseb1(n,b: positive integers with b>1)
2  v := n
3  k := logb(n,b)+1
4  for i := 1 to k
5     a_{k-i} := 0
6     while v ≥ b^{k-i}
7        a_{k-i} := a_{k-i}+1
8        v := v - b^{k-i}
9  return (a_{k-1},...,a_0){(a_{k-1}...a_0)_b is the base b expansion of n}
```

Least significant first: Start with right-most coefficient of expansion

Idea: (when $k > 1$)

$$\begin{aligned} n &= a_{k-1}b^{k-1} + \cdots + a_1 b + a_0 \\ &= b(a_{k-1}b^{k-2} + \cdots + a_1) + a_0 \end{aligned}$$

so $a_0 = n$ **mod** b and $a_{k-1}b^{k-2} + \cdots + a_1 = n$ **div** b.

Calculating base b expansion, from right

```
1  procedure baseb2(n,b: positive integers with b>1)
2  q := n
3  k := 0
4  while q ≠ 0
5     a_k := q mod b
6     q := q div b
7     k := k+1
8  return (a_{k-1},...,a_0){(a_{k-1}...a_0)_b is the base b expansion of n}
```

Base expansion review

Find and fix any and all mistakes with the following:

(a) $(1)_2 = (1)_8$

(b) $(142)_{10} = (142)_{16}$

(c) $(20)_{10} = (10100)_2$

(d) $(35)_8 = (1D)_{16}$

Base conversion algorithm

Recall the definition of base expansion we discussed:

Definition For b an integer greater than 1 and n a positive integer, the **base b expansion of n** is

$$(a_{k-1} \cdots a_1 a_0)_b$$

where k is a positive integer, $a_0, a_1, \ldots, a_{k-1}$ are nonnegative integers less than b, $a_{k-1} \neq 0$, and

$$n = a_{k-1} b^{k-1} + \cdots + a_1 b + a_0$$

Notice: *The base b expansion of a positive integer n is a string over the alphabet $\{x \in \mathbb{N} \mid x < b\}$ whose leftmost character is nonzero.*

Base b	Collection of possible coefficients in base b expansion of a positive integer
Binary ($b = 2$)	$\{0, 1\}$
Ternary ($b = 3$)	$\{0, 1, 2\}$
Octal ($b = 8$)	$\{0, 1, 2, 3, 4, 5, 6, 7\}$
Decimal ($b = 10$)	$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
Hexadecimal ($b = 16$)	$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$ letter coefficient symbols represent numerical values $(A)_{16} = (10)_{10}$ $(B)_{16} = (11)_{10}$ $(C)_{16} = (12)_{10}$ $(D)_{16} = (13)_{10}$ $(E)_{16} = (14)_{10}$ $(F)_{16} = (15)_{10}$

We write an algorithm for converting from base b_1 expansion to base b_2 expansion:

Fixed width definition

Definition For b an integer greater than 1, w a positive integer, and n a nonnegative integer _____, the **base** b **fixed-width** w **expansion of** n is

$$(a_{w-1} \cdots a_1 a_0)_{b,w}$$

where $a_0, a_1, \ldots, a_{w-1}$ are nonnegative integers less than b and

$$n = a_{w-1} b^{w-1} + \cdots + a_1 b + a_0$$

Fixed width example

Decimal $b = 10$	Binary $b = 2$	Binary fixed-width 10 $b = 2$, $w = 10$	Binary fixed-width 7 $b = 2$, $w = 7$	Binary fixed-width 4 $b = 2$, $w = 4$
$(20)_{10}$	(a)	(b)	(c)	(d)

Fixed width fractional definition

Definition For b an integer greater than 1, w a positive integer, w' a positive integer, and x a real number the **base b fixed-width expansion of x with integer part width w and fractional part width w'** is $(a_{w-1} \cdots a_1 a_0 . c_1 \cdots c_{w'})_{b,w,w'}$ where $a_0, a_1, \ldots, a_{w-1}, c_1, \ldots, c_{w'}$ are nonnegative integers less than b and

_____

and

_____

3.75 in fixed-width binary, integer part width 2, fractional part width 8	
0.1 in fixed-width binary, integer part width 2, fractional part width 8	

```
|welcome $jshell
|  Welcome to JShell -- Version 10.0.1
|  For an introduction type: /help intro

jshell> 0.1
$1 ==>

jshell> 0.2
$2 ==>

jshell> 0.1 + 0.2
$3 ==>

jshell> Math.sqrt(2)
$4 ==>

jshell> Math.sqrt(2)*Math.sqrt(2)
$5 ==>

jshell>
```

Note: Java uses floating point, not fixed width representation, but similar rounding errors appear in both.

Expansion summary

base b expansion of n	**base b fixed-width w expansion of n**
For b an integer greater than 1 and n a positive integer, the **base b expansion of n** is $(a_{k-1} \cdots a_1 a_0)_b$ where k is a positive integer, $a_0, a_1, \ldots, a_{k-1}$ are nonnegative integers less than b, $a_{k-1} \neq 0$, and $n = a_{k-1}b^{k-1} + \cdots + a_1 b + a_0$	For b an integer greater than 1, w a positive integer, and n a nonnegative integer with $n < b^w$, the **base b fixed-width w expansion of n** is $(a_{w-1} \cdots a_1 a_0)_{b,w}$ where $a_0, a_1, \ldots, a_{w-1}$ are nonnegative integers less than b and $n = a_{w-1}b^{w-1} + \cdots + a_1 b + a_0$

Negative int expansions

Representing negative integers in binary: Fix a positive integer width for the representation w, $w > 1$.

	To represent a positive integer n	To represent a negative integer $-n$
Sign-magnitude	$[0a_{w-2} \cdots a_0]_{s,w}$, where $n = (a_{w-2} \cdots a_0)_{2,w-1}$ Example $n = 17$, $w = 7$:	$[1a_{w-2} \cdots a_0]_{s,w}$, where $n = (a_{w-2} \cdots a_0)_{2,w-1}$ Example $-n = -17$, $w = 7$:
2s complement	$[0a_{w-2} \cdots a_0]_{2c,w}$, where $n = (a_{w-2} \cdots a_0)_{2,w-1}$ Example $n = 17$, $w = 7$:	$[1a_{w-2} \cdots a_0]_{2c,w}$, where $2^{w-1} - n = (a_{w-2} \cdots a_0)_{2,w-1}$ Example $-n = -17$, $w = 7$:
Extra example: 1s complement	$[0a_{w-2} \cdots a_0]_{1c,w}$, where $n = (a_{w-2} \cdots a_0)_{2,w-1}$ Example $n = 17$, $w = 7$:	$[1\bar{a}_{w-2} \cdots \bar{a}_0]_{1c,w}$, where $n = (a_{w-2} \cdots a_0)_{2,w-1}$ and we define $\bar{0} = 1$ and $\bar{1} = 0$. Example $-n = -17$, $w = 7$:

Calculating 2s complement

For positive integer n, to represent $-n$ in 2s complement with width w,

- Calculate $2^{w-1} - n$, convert result to binary fixed-width $w-1$, pad with leading 1, or
- Express $-n$ as a sum of powers of 2, where the leftmost 2^{w-1} is negative weight, or
- Convert n to binary fixed-width, flip bits, add 1 (ignore overflow)

Challenge: use definitions to explain why each of these approaches works.

Representing zero

Representing 0:

So far, the definitions of base expansions treat positive and negative integers. What about 0?

	To represent a **non-negative** integer n	To represent a **non-positive** integer $-n$
Sign-magnitude	$[0a_{w-2}\cdots a_0]_{s,w}$, where $n = (a_{w-2}\cdots a_0)_{2,w-1}$ Example $n = 0$, $w = 7$: (a)	$[1a_{w-2}\cdots a_0]_{s,w}$, where $n = (a_{w-2}\cdots a_0)_{2,w-1}$ Example $-n = 0$, $w = 7$: (b)
2s complement	$[0a_{w-2}\cdots a_0]_{2c,w}$, where $n = (a_{w-2}\cdots a_0)_{2,w-1}$ Example $n = 0$, $w = 7$: (c)	$[1a_{w-2}\cdots a_0]_{2c,w}$, where $2^{w-1} - n = (a_{w-2}\cdots a_0)_{2,w-1}$ Example $-n = 0$, $w = 7$: (d)

Fixed width addition

Fixed-width addition: adding one bit at time, using the usual column-by-column and carry arithmetic, and dropping the carry from the leftmost column so the result is the same width as the summands. *Does this give the right value for the sum?*

$$\begin{array}{r} (1\ 1\ 0\ 1\ 0\ 0)_{2,6} \\ +(0\ 0\ 0\ 1\ 0\ 1)_{2,6} \\ \hline \end{array}$$

$$\begin{array}{r} [1\ 1\ 0\ 1\ 0\ 0]_{s,6} \\ +[0\ 0\ 0\ 1\ 0\ 1]_{s,6} \\ \hline \end{array}$$

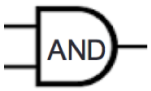
$$\begin{array}{r} [1\ 1\ 0\ 1\ 0\ 0]_{2c,6} \\ +[0\ 0\ 0\ 1\ 0\ 1]_{2c,6} \\ \hline \end{array}$$

Circuits basics

In a **combinatorial circuit** (also known as a **logic circuit**), we have **logic gates** connected by **wires**. The inputs to the circuits are the values set on the input wires: possible values are 0 (low) or 1 (high). The values flow along the wires from left to right. A wire may be split into two or more wires, indicated with a filled-in circle (representing solder). Values stay the same along a wire. When one or more wires flow into a gate, the output value of that gate is computed from the input values based on the gate's definition table. Outputs of gates may become inputs to other gates.

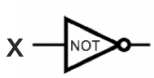
Logic gates definitions

Inputs		Output
x	y	x AND y
1	1	1
1	0	0
0	1	0
0	0	0



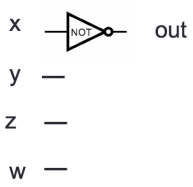
Inputs		Output
x	y	x XOR y
1	1	0
1	0	1
0	1	1
0	0	0

Input	Output
x	NOT x
1	0
0	1



Digital circuits basic examples

Example digital circuit:



Output when $x = 1, y = 0, z = 0, w = 1$ is _____
Output when $x = 1, y = 1, z = 1, w = 1$ is _____
Output when $x = 0, y = 0, z = 0, w = 1$ is _____

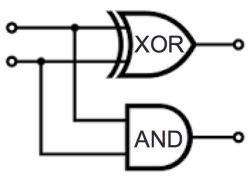
Draw a logic circuit with inputs x and y whose output is always 0. *Can you use exactly 1 gate?*

Half adder circuit

Fixed-width addition: adding one bit at time, using the usual column-by-column and carry arithmetic, and dropping the carry from the leftmost column so the result is the same width as the summands. In many cases, this gives representation of the correct value for the sum when we interpret the summands in fixed-width binary or in 2s complement.

For single column:

Input		Output	
x_0	y_0	s_0	c_0
1	1		
1	0		
0	1		
0	0		



Two bit adder circuit

Draw a logic circuit that implements fixed-width 2 binary addition:

- Inputs x_0, y_0, x_1, y_1 represent $(x_1x_0)_{2,2}$ and $(y_1y_0)_{2,2}$
- Outputs z_0, z_1, z_2 represent $(z_2z_1z_0)_{2,3} = (x_1x_0)_{2,2} + (y_1y_0)_{2,2}$ (may require up to width 3)

First approach: half-adder for each column, then combine carry from right column with sum of left column

Write expressions for the circuit output values in terms of input values:

$z_0 =$ _____

$z_1 =$ _____

$z_2 =$ _____

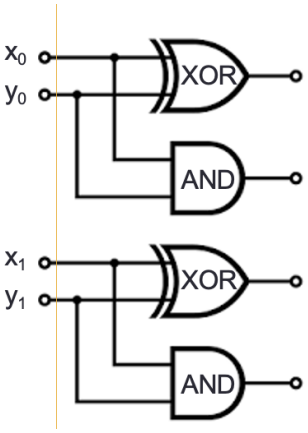
Second approach: for middle column, first add carry from right column to x_1, then add result to y_1

Write expressions for the circuit output values in terms of input values:

$z_0 =$ _____

$z_1 =$ _____

$z_2 =$ _____



Extra example Describe how to generalize this addition circuit for larger width inputs.