Week 2 at a glance

We will be learning and practicing to:

- Model systems with tools from discrete mathematics and reason about implications of modelling choices. Explore applications in CS through multiple perspectives, including software, hardware, and theory.
  - Selecting and representing appropriate data types and using notation conventions to clearly communicate choices
  - Determining the properties of positional number representations, including overflow and bit operations
- Translate between different representations to illustrate a concept.
  - Translating between symbolic and English versions of statements using precise mathematical language
  - Tracing algorithms specified in pseudocode
  - Representing numbers using positional representations, including decimal, binary, hexadecimal, fixed-width representations, and 2s complement
- Use precise notation to encode meaning and present arguments concisely and clearly
  - Precisely describing a set using appropriate notation e.g. roster method, set builder notation, and recursive definitions
  - Defining functions using multiple representations
- Know, select and apply appropriate computing knowledge and problem-solving techniques. Reason about computation and systems. Use mathematical techniques to solve problems. Determine appropriate conceptual tools to apply to new situations. Know when tools do not apply and try different approaches. Critically analyze and evaluate candidate solutions.
  - Using a recursive definition to evaluate a function or determine membership in a set
  - Using the definitions of the div and mod operators on integers

TODO:

#FinAid Assignment on Canvas (complete as soon as possible)

Review quiz based on Week 1 class material (due Monday 01/12/2026)

Homework assignment 1 (due Thursday 01/15/2026)

Review quiz based on Week 2 class material (due Monday 01/19/2026)

RQ1: Introduction and data types

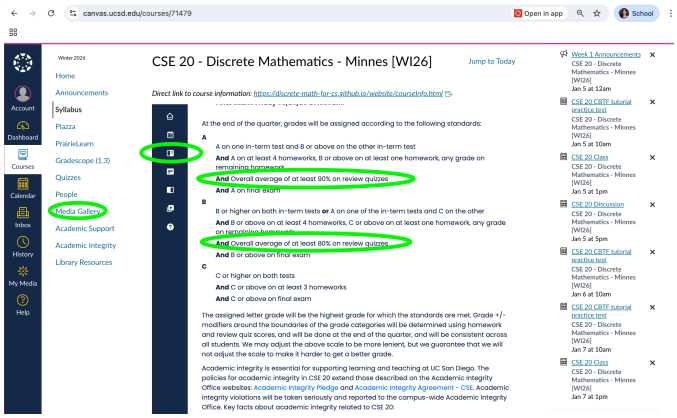
Total points: 0/20

0%

Assessment is **open** and you can answer questions.
Available credit: 100% until 23:59, Mon, Jan 12

This review quiz covers the material in Week 1 of the class. The review quizzes are due weekly, and are available on PrairieLearn. You can work towards full credit on a quiz by submitting (multiple attempts) up until the deadline; additional submissions after the original deadline are available for a limited time and for reduced credit. Review quiz questions are open to collaboration with everyone in CSE 20. Your goal should be to complete the review quiz questions as thoroughly as you need to make sure you can demonstrate the associated skills independently.

Question	Value	Variant history	Awarded points
Modeling			
RQ1.1. Defining colors using RGB representations	3		— /3
RQ1.2. Representing ratings as tuples	3		— /3
Sets and functions			
RQ1.3. Determining set membership	2		— /5
RQ1.4. Recursive definitions of sets	2		— /2
RQ1.5. Set operations with sets of RNA strands	4		— /4
RQ1.6. Defining functions: domain and codomain	3		— /3



Domain ? Codomain ?

Week 2 Monday: Sets, functions, and algorithms

Let's practice with functions related to some of our applications so far.

Recall: We model the collection of user ratings of the four movies Superman, Wicked, KPop Demon Hunters, A Minecraft Movie as the set $\{-1, 0, 1\}^4$. One function that compares pairs of ratings is

given by $d_0 : \{-1,0,1\}^4 \times \{-1,0,1\}^4 \to \mathbb{R}$ (Cartesian Product; = domain; codomain)

$$d_0(\ (\ (x_1, x_2, x_3, x_4), (y_1, y_2, y_3, y_4)\ )\ ) = \sqrt{(x_1-y_1)^2 + (x_2-y_2)^2 + (x_3-y_3)^2 + (x_4-y_4)^2}$$ Rule.

(ordered pair of 4-tuples)

Notice: any ordered pair of ratings is an okay input to d_0.

Notice: there are (at most)

$$(3 \cdot 3 \cdot 3 \cdot 3) \cdot (3 \cdot 3 \cdot 3 \cdot 3) = 3^8 = 6561$$

many pairs of ratings. There are therefore lots and lots of real numbers that are not the output of d_0.

How many inputs to d_0 are there?

$3 \times 3 \times 3 \times 3$ $\times$ $3 \times 3 \times 3 \times 3$

((-1/1, -1/1, -1/1, -1/1), ( ))

81 count $\times$ 81 count

Recall: RNA is made up of strands of four different bases that encode genomic information in specific ways. The bases are elements of the set $B = \{\mathtt{A}, \mathtt{C}, \mathtt{U}, \mathtt{G}\}$. 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 $sb \in S$

For $b \in B$, $b \in S$

If $s \in S$ and $b \in B$, then $sb \in S$.

where sb is string concatenation.

$S = \{ A, C, U, G, AA, AC, AU, AG, CA, CC, CU, CG, \ldots \}$

Note: $B \circ S = ?$ until we define S. Once we have S's definition, we can prove $B \circ S = S$.

Pro-tip: informal definitions sometime use $\cdots$ to indicate "continue the pattern". Often, to make this pattern precise we use recursive definitions.

Name	Domain	Codomain	Rule	Example
rnalen	S	$\mathbb{Z}^+$	Basis Step: If $b \in B$ then $rnalen(b) = 1$ Recursive Step: If $s \in S$ and $b \in B$, then $rnalen(sb) = 1 + rnalen(s)$	$rnalen(\texttt{AC}) \overset{\text{rec step}}{=} 1 + rnalen(\texttt{A})$ $\overset{\text{basis step}}{=} 1 + 1 = 2$
basecount	$S \times B$	$\mathbb{N}$	Basis Step: If $b_1 \in B, b_2 \in B$ then $basecount(\ (b_1, b_2)\ ) =$ $\begin{cases} 1 & \text{when } b_1 = b_2 \\ 0 & \text{when } b_1 \neq b_2 \end{cases}$ Recursive Step: If $s \in S, b_1 \in B, b_2 \in B$ $basecount(\ (sb_1, b_2)\ ) =$ $\begin{cases} 1 + basecount(\ (s, b_2)\ ) & \text{when } b_1 = b_2 \\ basecount(\ (s, b_2)\ ) & \text{when } b_1 \neq b_2 \end{cases}$	$basecount(\ (\texttt{ACU}, \texttt{C})\ ) =$ $= basecount(\ (\texttt{AC}, \texttt{C})\ )$ $= 1 + basecount(\ (\texttt{A}, \texttt{C})\ )$ $= 1 + 0$ $= 1$
"2 to the power of"	$\mathbb{N}$	$\mathbb{N}$	Basis Step: $2^0 = 1$ Recursive Step: If $n \in \mathbb{N}, 2^{n+1} = 2 \cdot 2^n$	$2^3 = 2^{2+1}$ $= 2 \cdot 2^2$ $= 2 \cdot 2^{1+1}$ $= 2 \cdot (2 \cdot 2^1)$ $= 2 \cdot (2 \cdot 2^{0+1})$ $= 2 \cdot (2 \cdot (2 \cdot 2^0))$ $= 2 \cdot (2 \cdot (2 \cdot 1))$ $= 2 \cdot 2 \cdot 2 \cdot 1 = 8$
"b to the power of i"	$\mathbb{Z}^+ \times \mathbb{N}$	$\mathbb{N}$	Basis Step: $b^0 = 1$ Recursive Step: If $i \in \mathbb{N}, b^{i+1} = b \cdot b^i$	

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

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.

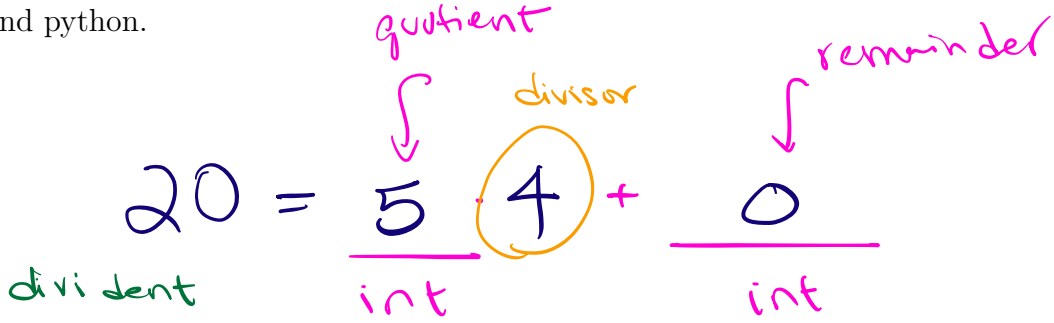
Because these numbers are guaranteed to exist, the following functions are well-defined:

div : $\mathbb{Z} \times \mathbb{Z}^+ \to \mathbb{Z}$ given by **div** ((n, d)) is the quotient when n is the dividend and d is the divisor.

mod : $\mathbb{Z} \times \mathbb{Z}^+ \to \mathbb{Z}$ given by **mod** ((n, d)) is the remainder when n is the dividend and d is the divisor.

Because these functions are so important, we sometimes use the notation n **div** $d =$ **div** ((n, d)) and n **mod** $d =$ **mod** ((n, d)).

Pro-tip: The functions **div** and **mod** are similar to (but not exactly the same as) the operators / and % in Java and python.



Example calculations:

20 **div** 4 = 5

20 **mod** 4 = 0

because $20 = 5 \cdot 4 + 0$

20 **div** 3 = 6

20 **mod** 3 = 2

because $20 = 6 \cdot 3 + 2$

-20 **div** 3 = -7

-20 **mod** 3 = 1

because $-20 = -7 \cdot 3 + 1$

remainder must be nonnegative

Week 2 Wednesday: Representing 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?

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

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

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

$$n = \sum_{i=0}^{k-1} a_i b^i = a_0 b^0 + \cdots + a_{k-1} \cdot b^{k-1}$$

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 $(A)_{16} = (10)_{10}$ $(B)_{16} = (11)_{10}$ $(C)_{16} = (12)_{10}$ $(D)_{16} = (13)_{10}$ $(E)_{16} = (14)_{10}$ $(F)_{16} = (15)_{10}$

Examples:

~~$(1401)_2$~~ Base 2 expansion can't use symbol 4!

$(1401)_{10} = 1\cdot 10^3 + 4\cdot 10^2 + 0\cdot 10^1 + 1\cdot 10^0$

10^0, 10^1, 10^2, 10^3

= One thousand four hundred and one.

$(1401)_{16} = 1\cdot 16^3 + 4\cdot 16^2 + 0\cdot 16^1 + 1\cdot 16^0 = 4096 + 4\cdot 256 + 1 = 5121$

$16^0 = 1$, $16^1 = 16$, $16^2 = 256$, $16^3 = 4096$

$= (5121)_{10}$

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

Algorithms can be expressed in English or in more formalized descriptions like pseudocode or fully executable programs.

Sometimes, we can define algorithms whose output matches the rule for a function we already care about. Consider the (integer) logarithm function

$$logb : \{b \in \mathbb{Z} \mid b > 1\} \times \mathbb{Z}^+ \to \mathbb{N}$$

codomain

Cartesian product of the set of integers greater than 1 with the set of positive integers

defined by

rule

$$logb(\ (b, n)\ ) = \text{greatest integer } y \text{ so that } b^y \text{ is less than or equal to } n$$

Calculating integer part of base b logarithm

```
1  procedure logb(b,n: positive integers with b>1)
2  i := 0
3  while n > b-1
4    i := i+1
5    n := n div b
6  return i {i holds the integer part of the base b logarithm of n}
```

Trace this algorithm with inputs $b = 3$ and $n = 17$

	b	n	i	$n > b-1$?
Initial value	3	17	0	T
After 1 iteration		5	1	T
After 2 iterations		1	2	F
After 3 iterations				

17 > 2, 5 > 2, 1 > 2

17 div 3 = 5, 5 div 3

$17 = 3\cdot 5 + 2$, $5 = 3\cdot 1 + 2$

$logb((3, 17)) = 2$

$3^0 = 1$, $3^1 = 3$, $3^2 = 9$, $3^3 = 27$

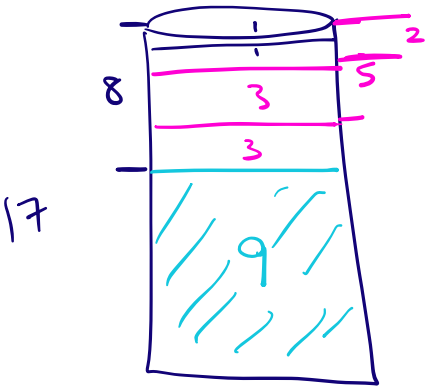
Compare: does the output match the rule for the (integer) logarithm function? Yes!

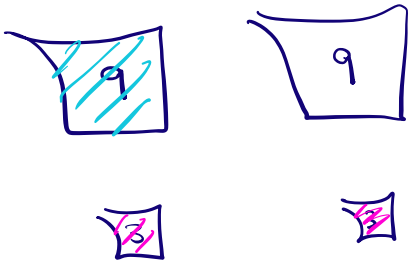
2 = 2

Two algorithms for constructing base b expansion from decimal representation

Most significant first: Start with left-most coefficient of expansion (highest value)

Informally: Build up to the value we need to represent in "greedy" approach, using units determined by base.





$17 = 1\cdot 3^2 + 2\cdot 3^1 + 2\cdot 3^0 = (122)_3$

Calculating base b expansion, from left

```
1  procedure basebmost(n,b: positive integers with b>1)
2  v := n                                  remaining value
3  k := 1+ output of logb algorithm with inputs b and n
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}
```

related to biggest measuring cup I can use

Trace with $n=17$, $b=3$.

	n	b	v	k	i	a_{k-i}	$v \geq b^{k-i}$?	
initial value	17	3	17	3	1			
$i=1$ in for loop			~~17~~ 8		1	$a_2 = $ ~~0~~ 1	$17 \geq 3^2$? $8 \geq 3^2$?	T F
$i=2$ in for loop			~~8~~ ~~5~~ 2		2	$a_1 = $ ~~0~~ 2	$8 \geq 3^1$? $5 \geq 3^1$? $2 \geq 3^1$?	T T F
$i=3$ in for loop			~~2~~ ~~1~~ 0		3	$a_0 = $ ~~0~~ 2	$2 \geq 3^0$? $1 \geq 3^0$? $0 \geq 3^0$?	T T F

index fixed after class

Return $(1,2,2)$ corresponding to $(122)_3$

Least significant first: Start with right-most coefficient of expansion (lowest value)

Idea: (when $k > 1$)

$$n = (a_{k-1}\ a_{k-2}\ \cdots\ a_1\ a_0)_b$$

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

Long division

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

Base 3 expansion of 17, using basebleast algorithm

$n = 17$ $b = 3$

result of div | result of mod

$17 = 5 \cdot 3 + 2$

$5 = 1 \cdot 3 + 2$

$1 = 0 \cdot 3 + 1$

	k	q	a_k	$q \neq 0$?
Initial?	0	17		T
After 1 iteration	1	5	$a_0 = 2$	T
After 2 iterations	2	1	$a_1 = 2$	T
After 3 iterations	3	0	$a_2 = 1$	F

Output (1, 2, 2) which corresponds to $(122)_3$ is the base 3 expansion of 17.

3^2 3^1 3^0

Confirm: $(122)_3 = 1 \cdot 9 + 2 \cdot 3 + 2 \cdot 1 = 9 + 6 + 2 = 17$:)

$(a_{k-1}\cdots a_0)_b = a_{k-1}b^{k-1} + \cdots + a_0 b^0$

Week 2 Friday: Algorithms for numbers

Find and fix any and all mistakes with the following: LHS = RHS ?

(a) $(1)_2 = (1)_8$ (2° = 8°)

(b) $(142)_{10} \neq (142)_{16}$

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

(d) $(35)_8 = (1D)_{16}$ (8's, 1's; 16's, 1's)

RHS $= 1\cdot 16^2 + 4\cdot 16^1 + 2\cdot 16^0 = 256 + 4\cdot 16 + 2 = 322$

Bonus: what is the hexadecimal expansion of 142?

RHS $= 1\cdot 2^4 + 0\cdot 2^3 + 1\cdot 2^2 + 0\cdot 2^1 + 0\cdot 2^0 = 16 + 4 = 20$ = LHS of (c) ☺

LHS $= 3\cdot 8 + 5\cdot 1 = 29$

RHS $= 1\cdot 16 + 13\cdot 1 = 29$ so LHS = RHS ☺

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

Given a base b_1 and an expansion $(a_{k-1}\cdots a_0)_{b_1}$ and a target base b_2 which is a positive integer greater than 1,

(1) Calculate $n = \sum_{i=0}^{k-1} a_i b_1^i$

(2) Use baseBleast algorithm with inputs n and b_2 to calculate base b_2 expansion of n

(3) Output this expansion.

fix the number of columns

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

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

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

$$n = \sum_{i=0}^{w-1} a_i b^i$$

eg. base 2

$(1\ 0)_2$

$(1\ 1\ 1)_2$

$(1\ 0\ 0\ 0)_2$

$(1\ 0\ 0\ 1)_2$

$(1\ 1\ 1\ 1\ 1\ 1\ \cdots\ 1)_2$

$(1\ 0\ 0\ \cdots\ \ \ 0)_2$

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 $b = 2$, $w = 4$
$(20)_{10}$	$(10100)_2$	$(0000010100)_{2,10}$	$(0010100)_{2,7}$	

$20 = 16 + 4 = 2^4 + 2^2$

$= 1\cdot 2^4 + 0\cdot 2^3 + 1\cdot 2^2 + 0\cdot 2^1 + 0\cdot 2^0$

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 $(a_{w-1}\cdots a_1a_0.c_1\cdots c_{w'})_{b,w,w'}$ where $a_0, a_1, \ldots, a_{w-1}, c_1, \ldots, c_{w'}$ are nonnegative integers less than b and

$$x \geq \sum_{i=0}^{w-1} a_i b^i + \sum_{j=1}^{w'} c_j b^{-j} \quad \text{and} \quad x < \sum_{i=0}^{w-1} a_i b^i + \sum_{j=1}^{w'} c_j b^{-j} + b^{-w'}$$

approximate!

3.75 in fixed-width binary, integer part width 2, fractional part width 8	$(1\,1.1\,1\,0\,0\,0\,0\,0\,0)_{2,2,8}$ $2^{-1} = \frac{1}{2} = 0.5$ $2^{-2} = \frac{1}{4} = 0.25$
0.1 in fixed-width binary, integer part width 2, fractional part width 8	$(0\,0.0\,0\,0\,1\,1\,0\,0\,1)_{2,2,8}$

$\frac{1}{2} = 0.5$

$\frac{1}{4} = 0.25$

$\frac{1}{8} = 0.125$

$\frac{1}{16} = 0.0625$

$\frac{1}{32} = 0.03125$

$\frac{1}{64} = 0.015625$

$\frac{1}{128} = 0.0078125$

$\frac{1}{256} = 0.00390625$

$0.1 - 0.0625 = 0.03755$

$0.03755 - 0.03125 = 0.0063$

$0.0063 - 0.00390625 = 0.00239375$

approximation error in this representation

```
[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.

We'll do the rest in Week 3.
The deadline for full credit for RQ2 has been extended by a week accordingly.

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 $n = 17$, $w = 7$: $17 = [0\ 0\ 1\ 0\ 0\ 0\ 1]_{s,7}$ (SIGN, 32, 16, 8, 4, 2, 1) $17 = 0 \cdot 32 + 1 \cdot 16 + 0 \cdot 8 + 0 \cdot 4 + 0 \cdot 2 + 1 \cdot 1$	$[1a_{w-2} \cdots a_0]_{s,w}$, where $n = (a_{w-2} \cdots a_0)_{2,w-1}$ Example $-n = -17$, $w = 7$: $-17 = [1\ 0\ 1\ 0\ 0\ 0\ 1]_{s,7}$ (SIGN, 32, 16, 8, 4, 2, 1) $17 = 1 \cdot 16 + 0 \cdot 8 + 0 \cdot 4 + 0 \cdot 2 + 1 \cdot 1$
2s complement	$[0a_{w-2} \cdots a_0]_{2c,w}$, where $n = (a_{w-2} \cdots a_0)_{2,w-1}$ Example $n = 17$, $w = 7$: $17 = [0\ 0\ 1\ 0\ 0\ 0\ 1]_{2c,7}$ (-64, 32, 16, 8, 4, 2, 1) $17 = 0 \cdot 32 + 1 \cdot 16 + 0 \cdot 8 + 0 \cdot 4 + 0 \cdot 2 + 1 \cdot 1$	$[1a_{w-2} \cdots a_0]_{2c,w}$, where $2^{w-1} - n = (a_{w-2} \cdots a_0)_{2,w-1}$ Example $-n = -17$, $w = 7$: $-17 = [1\ 1\ 0\ 1\ 1\ 1\ 1]_{2c,7}$ (-64, 32, 16, 8, 4, 2, 1) $17 = 1 \cdot (-64) + 1 \cdot 32 + 0 \cdot 16 + 1 \cdot 8 + 1 \cdot 4 + 1 \cdot 2 + 1 \cdot 1$

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 w, flip bits, add 1 (ignore overflow)

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

Representing 0:

So far, we have representations for 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 $n = 0$, $w = 7$: $0 = [0\ 0\ 0\ 0\ 0\ 0\ 0]_{s,7}$ (SIGN, 32, 16, 8, 4, 2, 1)	$[1a_{w-2}\cdots a_0]_{s,w}$, where $n = (a_{w-2}\cdots a_0)_{2,w-1}$ Example $-n = 0$, $w = 7$: $0 = [1\ 0\ 0\ 0\ 0\ 0\ 0]_{s,7}$ (SIGN, 32, 16, 8, 4, 2, 1)
2s complement	$[0a_{w-2}\cdots a_0]_{2c,w}$, where $n = (a_{w-2}\cdots a_0)_{2,w-1}$ Example $n = 0$, $w = 7$: $0 = [0\ 0\ 0\ 0\ 0\ 0\ 0]_{2c,7}$ (32, 16, 8, 4, 2, 1) UNIQUE REPRESENTATION OF ZERO IN 2's COMPLEMENT	$[1a_{w-2}\cdots a_0]_{2c,w}$, where $2^{w-1} - n = (a_{w-2}\cdots a_0)_{2,w-1}$ Example $-n = 0$, $w = 7$: $2^{w-1} - n = (a_{w-2}\cdots a_0)_{2,w-1}$ $2^{7-1} - 0 = 2^6 = 64$ ($w-1=6$) but 64 has no base 2 fixed width 6 expansion

With 3 bit strings are 000, 001, 010, 011, 100, 101, 110, 111

Used as base 2 fixed width 3:

$(000)_{2,3}=0$, $(001)_{2,3}=1$, $(010)_{2,3}=2$, $(011)_{2,3}=3$, $(100)_{2,3}=4$, $(101)_{2,3}=5$, $(110)_{2,3}=6$, $(111)_{2,3}=7$

Used as sign-magnitude width 3.

$[111]_{s,3}=-3$, $[110]_{s,3}=-2$, $[101]_{s,3}=-1$, $[100]_{s,3}=0$, $[000]_{s,3}=0$, $[001]_{s,3}=1$, $[010]_{s,3}=2$, $[011]_{s,3}=3$

Used as 2s complement width 3

$[100]_{2,3}=-4$, $[101]_{2,3}=-3$, $[110]_{2,3}=-2$, $[111]_{2,3}=-1$, $[000]_{s,3}=0$, $[001]_{s,3}=1$, $[010]_{s,3}=2$, $[011]_{s,3}=3$

