The MathGod.org Human Curriculum

Volume K: Kindergarten Math Workbook

A 3-Level Illustrated Edition (Big Text · Bridge to Lean · Pure Code to Assembly)

MathGod.org Open Curriculum Collective

Built alongside OpenStax & Lean 4

Public Domain · Free for All, Forever

Dedication

"Math teachers at local in-person schools are the only way to truly learn math. Online, Khan Academy is more established and well-resourced than us, and we recommend that you check it out. But here at mathgod.org we publish, free for all forever, a full human-oriented math textbook based on standard curriculum we, the authors, experienced ourselves since childhood. We use OpenStax, and thank them for their quality effort transcending the pecuniary realities of work. Our textbook has 3 levels. The top level is reading and demonstrations. The middle level is clickable-at-all-points extra info and the bridge to Lean code, i.e., how computers do math (and we may deviate from Lean, using other computer languages, for some lessons when it benefits clarity). The bottom level is pure code (which can go as deep as assembly-language in some parts). Why are we doing this? Because math should be shared."

Contents

1 K.1 Counting Starts at 1 (and the Secret of 0) **2**

2 K.2 The Magic Word "Next!" (Successor) **3**

3 K.3 Counting to 3 & K.4 Counting to 5 (One Full Hand) **4**

4 K.5 Counting to 10 (Two Hands & The Ten-Frame) **4**

5 K.6 Putting Groups Together: Addition (2 + 3 = 5) 6

6 K.7 Taking Away: Subtraction & The Impossible Cookie 6

7 K.8 Shapes All Around Us: Circle, Triangle, Square & Hexagon 7

1 K.1 Counting Starts at 1 (and the Secret of 0)

[LEVEL 1: BIG-TEXT READING & PICTURE WORKBOOK]

Hold up ONE finger!

When humans count things we can touch—like shiny red apples—we always start at **ONE (1)**.



0 Apples **1 Apple** **2 Apples**

One (1) means something is really there! What if the apple basket is completely empty? We call an empty basket **ZERO (0)**.

[LEVEL 2: EXTRA INFO & BRIDGE TO COMPUTER MATH]

Yin and Yang of Counting: Why do Kindergarteners start counting at 1, while computers start counting at 0? Humans count *presence* (one apple, one finger), whereas a digital computer begins as an unpowered silicon register with all switches off ($00000000_2 = 0$, the Machine Void). In Lean 4, natural numbers $\mathbb{N}$ begin at `Nat.zero` (0), and human 1 is `Nat.succ`$(0) = 1$.

[LEVEL 3: PURE CODE — LEAN 4, PYTHON & ARM64 ASSEMBLY]

```
-- 1. Lean 4 Formalization (Peano Arithmetic)
abbrev void_zero : Nat := 0
abbrev human_one : Nat := Nat.succ void_zero
theorem human_starts_from_one : human_one = 1 := by rfl

# 2. Python Clarity View
basket = []          # 0 apples (the machine void)
basket.append("apple")
assert len(basket) == 1

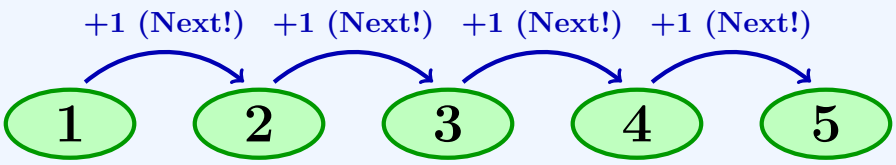
; 3. ARM64 Assembly Language (CPU Silicon Register X0)
EOR X0, X0, X0       ; Clear register X0 to 0 (00000000_2: Machine Void)
ADD X0, X0, #1       ; Step once into existence: X0 = 1 (00000001_2)
```

2 K.2 The Magic Word "Next!" (Successor)

[LEVEL 1: BIG-TEXT READING & PICTURE WORKBOOK]

How do we get to the NEXT number?

Imagine a little green frog hopping across lily pads on a pond. Each time you clap your hands and say **"NEXT!"**, the frog hops **1 step forward**!



Because you can *always* take one more hop, the numbers never, ever end!

[LEVEL 2: EXTRA INFO & BRIDGE TO COMPUTER MATH]

The Successor Function $S(n) = n+1$: When a child asks "What is the biggest number?", the answer is that there isn't one—because any number n has a **successor** $\text{succ}(n) = n+1$. In Lean 4, `Nat.succ` is the single constructor that builds the infinite universe of counting numbers.

[LEVEL 3: PURE CODE — LEAN 4, PYTHON & ARM64 ASSEMBLY]

```
-- 1. Lean 4: Every hop is Nat.succ
abbrev two   : Nat := Nat.succ 1
abbrev three : Nat := Nat.succ two
theorem next_after_two_is_three : Nat.succ 2 = 3 := by rfl

# 2. Python Clarity Function
def next_number(n: int) -> int:
    return n + 1

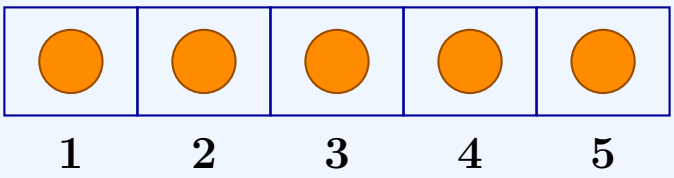
; 3. x86-64 / ARM64 Hardware Increment Instruction
MOV RAX, 2           ; Load 2 into accumulator register RAX
INC RAX              ; Single-cycle hardware successor: RAX becomes 3
```

3 K.3 Counting to 3 & K.4 Counting to 5 (One Full Hand)

[LEVEL 1: BIG-TEXT READING & PICTURE WORKBOOK]

1 Sun, 2 Eyes, 3 Triangle Corners, 5 Fingers!

Look at a **Five-Frame** box. When we place **5 golden stars** inside, every single box is full—just like all **5 fingers** on one hand!



When one box has **5 stars** and another has **3 stars**, 5 is **MORE** than 3 ($5 > 3$)!

[LEVEL 2: EXTRA INFO & BRIDGE TO COMPUTER MATH]

Subitizing & Order ($a > b$): Human brains instantly recognize quantities of $1, 2, 3$ without counting one-by-one (called *subitizing*). A 5-Frame anchors numbers to our five fingers. In Lean 4 and OpenStax Prealgebra Section 1.1, $5 > 3$ means 5 lies to the right of 3 on the number line ($3 + 2 = 5$).

[LEVEL 3: PURE CODE — LEAN 4, PYTHON & ARM64 ASSEMBLY]

```
-- 1. Lean 4: Counting to 5 and Comparing More
abbrev three : Nat := 3
abbrev five  : Nat := Nat.succ (Nat.succ three)
theorem five_is_three_plus_two : three + 2 = five := by rfl

# 2. Python Clarity View
assert 5 > 3 and (5 - 3) == 2

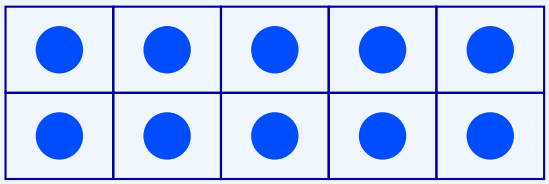
; 3. ARM64 Comparison & Status Flags (CMP)
MOV W0, #5           ; W0 = 5 (0101_2)
CMP W0, #3           ; Compute 5 - 3 = 2 > 0 (Sets Greater-Than condition)
```

4 K.5 Counting to 10 (Two Hands & The Ten-Frame)

[LEVEL 1: BIG-TEXT READING & PICTURE WORKBOOK]

Two Hands Make TEN (10)!

Stack two Five-Frames together and you get a **Ten-Frame**! Five on top plus five on the bottom makes **TEN (10)**.



$$5 + 5 = 10$$

Notice how we write **10**: the **1** means **1 full Ten-Frame**, and the **0** means **0 extra ones outside**!

[LEVEL 2: EXTRA INFO & BRIDGE TO COMPUTER MATH]

Base-10 (Human Fingers) vs. Base-2 (Computer Switches): Humans invented Base-10 (*decimal* place value, OpenStax Section 1.1) because we have 10 fingers: $10 = 1 \times 10^1 + 0 \times 10^0$. Computers only have 2 "fingers" (voltage ON = 1 and OFF = 0), so a computer writes ten in Base-2 (*binary*) as $1010_2 = 8 + 0 + 2 + 0 = 10$.

[LEVEL 3: PURE CODE — LEAN 4, PYTHON & ARM64 ASSEMBLY]

```
-- 1. Lean 4: Two Fives Make One Ten
theorem two_hands_make_ten : 5 + 5 = 10 := by rfl

# 2. Python Clarity View (Decimal vs Binary Base Conversion)
ten = 5 + 5
assert str(ten) == "10" and bin(ten) == "0b1010"

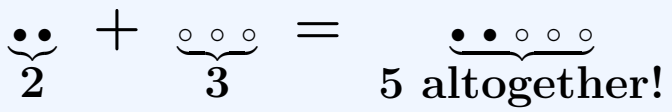
; 3. ARM64 Binary Representation in Silicon
MOV W0, #10          ; Stored in register bits as 00001010_2 (8 + 2 = 10)
```

5 K.6 Putting Groups Together: Addition ($2 + 3 = 5$)

[LEVEL 1: BIG-TEXT READING & PICTURE WORKBOOK]

Putting Two Groups Into One Big Basket!

You hold **2 red apples** in your left hand and **3 green apples** in your right hand. When we slide them together, we use the **PLUS SIGN** (+):



And what if you swap your hands? $3 + 2$ **is STILL** 5**!** What if you add an empty basket $(4 + 0)$? You still have 4!

[LEVEL 2: EXTRA INFO & BRIDGE TO COMPUTER MATH]

From Finger Counting to ALU Full Adders: In Lean 4, $a+b$ is defined recursively on the second argument: $a+0=a$, and $a+\text{succ}(b)=\text{succ}(a+b)$. Inside a CPU's Arithmetic Logic Unit (ALU), binary addition $0010_2 + 0011_2 = 0101_2$ happens in one nanosecond via XOR sum gates and AND carry gates!

[LEVEL 3: PURE CODE — LEAN 4, PYTHON & ARM64 ASSEMBLY]

```
-- 1. Lean 4 Formal Proof
theorem two_plus_three_is_five : 2 + 3 = 5 := by rfl
theorem flip_hands_same_total  : 2 + 3 = 3 + 2 := by rfl
theorem adding_zero_changes_nothing : 4 + 0 = 4 := by rfl

# 2. Python Clarity View
assert len(["red", "red"] + ["green", "green", "green"]) == 5

; 3. ARM64 Assembly Language (ALU Addition)
MOV W0, #2           ; W0 = 00000010_2 (2)
MOV W1, #3           ; W1 = 00000011_2 (3)
ADD W2, W0, W1       ; W2 = 00000101_2 (5) via hardware full-adder gates
```

6 K.7 Taking Away: Subtraction & The Impossible Cookie

[LEVEL 1: BIG-TEXT READING & PICTURE WORKBOOK]

Popping Balloons & Eating Cookies!

If you have **5 balloons** and **2 balloons POP**, you count backward two steps: $5 \to 4 \to 3$. So $5 - 2 = 3$!

The Cookie Mystery: What if there are only **2 cookies** on a plate, and a hungry monster tries to eat **9 cookies**? After eating the 2 cookies, the plate is empty! There are **0 cookies left**.

[LEVEL 2: EXTRA INFO & BRIDGE TO COMPUTER MATH]

Why $2 - 9 = 0$ in $\mathbb{N}$ (Saturating Subtraction) vs. -7 in OpenStax Ch. 3: In Kindergarten natural numbers $\mathbb{N} = \{0, 1, 2, \dots\}$, there are no negative cookies! Lean 4 defines natural subtraction via predecessor steps that stop at 0: $\texttt{pred}(0) = 0$, so $2 - 9 = 0$. Later, in **OpenStax Prealgebra Chapter 3 (Integers $\mathbb{Z}$)**, we invent negative numbers so $2 - 9 = -7$ (owing 7 cookies!).

[LEVEL 3: PURE CODE — LEAN 4, PYTHON & ARM64 ASSEMBLY]

```
-- 1. Lean 4: Natural Subtraction Saturates at Zero
theorem five_take_away_two : 5 - 2 = 3 := by rfl
theorem impossible_cookie_subtraction : 2 - 9 = 0 := by rfl

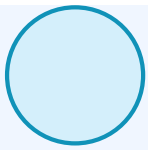
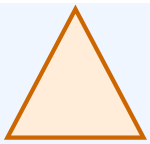
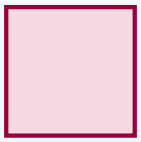
# 2. Python Clarity View (Kindergarten Floor vs Integer Debt)
def natural_sub(a: int, b: int) -> int:
    return max(0, a - b)   # Saturates at 0 on a physical plate!

; 3. ARM64 Assembly: Unsigned Saturating Subtraction via Carry Flag
MOV  W0, #2          ; 2 cookies on plate
SUBS W0, W0, #9      ; Subtract 9 and set CPU flags (C=0 means borrow)
CSEL W0, W0, WZR, CS ; If borrow occurred (C=0), clamp W0 to Zero (WZR=0)
```

7 K.8 Shapes All Around Us: Circle, Triangle, Square & Hexagon

[LEVEL 1: BIG-TEXT READING & PICTURE WORKBOOK]

Count the Straight Sides and Corners!

Circle (0 sides) **Triangle (3 sides)** **Square (4 sides)**

A **Circle** curves all the way around with **0 straight sides**. A **Triangle** has **3 straight sides**. A **Square** has **4 equal straight sides**!

[LEVEL 2: EXTRA INFO & BRIDGE TO COMPUTER MATH]

Polygons, Vertices & Graphics Meshes: Every closed flat polygon has the exact same number of straight sides (edges E) as sharp corners (vertices V): $V = E$. In 3D computer graphics and video games, GPUs build almost every curved surface in the world out of millions of tiny **triangles** (3 vertices), because any 3 points in space are guaranteed to lie on a flat plane!

[LEVEL 3: PURE CODE — LEAN 4, PYTHON & ARM64 ASSEMBLY]

```
-- 1. Lean 4: Counting Sides of Shapes
abbrev circle_sides   : Nat := 0
abbrev triangle_sides : Nat := 3
abbrev square_sides   : Nat := 4
theorem triangle_plus_one_is_square : triangle_sides + 1 = square_sides := by rfl

# 2. Python Clarity View
shapes = {"circle": 0, "triangle": 3, "square": 4, "hexagon": 6}
assert shapes["square"] - shapes["triangle"] == 1

; 3. ARM64 Assembly: Triangle + Square Total Sides (3 + 4 = 7)
MOV W0, #3           ; 3 triangle sides
ADD W0, W0, #4       ; + 4 square sides = 7 sides altogether
```