📚 Module 1: Number Fundamentals - Complete Notes

🎯 What You'll Learn

In this module, you'll master the building blocks of all programming: **counting, stepping through sequences, and understanding how numbers are structured**. This is the foundation of every loop, every array traversal, and every iteration you'll ever write.

📖 Concept Explained (Like a YouTube Video)

The Basics

Hey everyone! Welcome to Module 1. Today we're going to talk about something that sounds super basic but is **absolutely critical** for programming — numbers and how we count them.

Think about it: when you write a loop, what are you really doing? **You're counting!** From 0 to 10, from 10 down to 0, or maybe skipping every other number.

Let's break this down:

🔢 Counting Forward

Imagine you're climbing stairs. You start at step 0, then 1, 2, 3... You're **incrementing** by 1 each time.

```
Step 0 → Step 1 → Step 2 → Step 3 → Step 4 → ...
   ↑        +1       +1       +1       +1
 START
```

🔢 Counting Backward

Now imagine going back down. Start at the top (let's say step 5), then 4, 3, 2, 1, 0. **Decrementing** by 1 each time.

```
Step 5 → Step 4 → Step 3 → Step 2 → Step 1 → Step 0
   ↑        -1       -1       -1       -1       -1
 START                                         STOP
```

🔢 Skip Counting

What if you're in a hurry and skip steps? Taking 2 at a time: 0, 2, 4, 6, 8...

```
Step 0 → Step 2 → Step 4 → Step 6 → Step 8
   ↑        +2       +2       +2       +2
 START    (skip 1)  (skip 3) (skip 5) (skip 7)
```

Visual Understanding

Think of a **number line** like a ruler:

```
    ┌───┬───┬───┬───┬───┬───┬───┬───┬───┬───┬───┐
    │ 0 │ 1 │ 2 │ 3 │ 4 │ 5 │ 6 │ 7 │ 8 │ 9 │10 │
    └───┴───┴───┴───┴───┴───┴───┴───┴───┴───┴───┘

      ↑                                        ↑
```

```
   START                                    END

   Forward: Move RIGHT (add)
   Backward: Move LEFT (subtract)
   Skip: Jump by step size
```

Place Value System

Now here's something SUPER useful for programming — understanding **place value**.

Take the number **5847**:

```
   5    8    4    7
   ↓    ↓    ↓    ↓
 1000s 100s 10s  1s
```

Each position has a **weight**:

- 7 × 1 = 7
- 4 × 10 = 40
- 8 × 100 = 800
- 5 × 1000 = 5000
- Total: 5847

Why does this matter for coding? Because when you need to:

- Extract digits from a number
- Validate credit card numbers
- Parse numerical input
- Work with binary/hex

You'll use the same logic!

The Math Behind It

Concept	Rule	Example
Counting Forward	start at 0, add 1 repeatedly	0, 1, 2, 3, 4...
Counting Backward	start at n, subtract 1 repeatedly	5, 4, 3, 2, 1, 0
Skip Counting	start at 0, add k repeatedly	0, 3, 6, 9, 12... (k=3)
Place Value	digit × position_weight	7 in 5847 = 7 × 1 = 7
Extract rightmost digit	number % 10	5847 % 10 = 7
Remove rightmost digit	number // 10	5847 // 10 = 584

💻 Programming Connection

Why Programmers Need This

1. **Every `for` loop is counting** — You specify start, end, and step
2. **Array indexing is counting** — Arrays are numbered sequences

3. **Pagination is skip counting** — Pages of 10 items: 0, 10, 20, 30...

Code Examples

```
# Example 1: Counting Forward (0 to 9)
# Think: "I'm walking up stairs, one step at a time"

for i in range(10):
    print(f"Step {i}")

# Output: Step 0, Step 1, Step 2, ... Step 9

# 🔑 Key insight: range(10) means 0 to 9 (doesn't include 10!)
```

```
# Example 2: Counting Backward (10 to 1)
# Think: "I'm going back down, counting down like a rocket launch"

for i in range(10, 0, -1):
    print(f"Countdown: {i}")
print("🚀 Liftoff!")

# Output: 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 🚀 Liftoff!

# 🔑 Key insight: range(start, stop, step) - stop is excluded!
```

```
# Example 3: Skip Counting (Even numbers)
# Think: "I'm in a hurry, skipping every other step"

for i in range(0, 20, 2):
    print(f"Even number: {i}")

# Output: 0, 2, 4, 6, 8, 10, 12, 14, 16, 18

# 🔑 Key insight: step = 2 means skip by 2
```

```
# Example 4: Extract All Digits from a Number
# Think: "I'm peeling off digits from the right, like layers of an onion"

def extract_digits(number):
    digits = []
    while number > 0:
        digit = number % 10      # Get rightmost digit
        digits.append(digit)     # Save it
        number = number // 10    # Remove rightmost digit
    return digits[::-1]          # Reverse to get correct order

# Test it
print(extract_digits(5847))  # [5, 8, 4, 7]
```

```
# 🔑 Key insight:
# - % 10 gives you the last digit
# - // 10 removes the last digit
```

```
# Example 5: Sum of Digits (Classic programming problem!)
# Think: "Extract each digit, add to running total"

def sum_of_digits(n):
    total = 0
    while n > 0:
        total += n % 10  # Add rightmost digit
        n = n // 10      # Remove rightmost digit
    return total

print(sum_of_digits(5847))  # 5 + 8 + 4 + 7 = 24
```

Common Use Cases

1. **Iterating through arrays/lists** — `for i in range(len(array))`
2. **Generating test IDs** — `for i in range(1, 101)` → TC_001 to TC_100
3. **Pagination** — `for page in range(0, total, page_size)`
4. **Processing digits** — Credit card validation, ISBN checks
5. **Reverse iteration** — Processing from end to start

🧪 SDET/Testing Application

How This Helps in Testing

As an SDET, you'll use counting logic constantly:

1. **Generate sequential test IDs**
2. **Paginate through API responses**
3. **Process log files line by line**
4. **Create test data with patterns**

Real-World Testing Example

```
# SDET Scenario: Generate Test Case IDs

def generate_test_ids(module_name, count):
    """Generate formatted test case IDs"""
    test_ids = []
    for i in range(1, count + 1):
        # Zero-pad to 3 digits: 001, 002, ..., 100
        test_id = f"{module_name}_TC_{i:03d}"
        test_ids.append(test_id)
    return test_ids

# Generate 5 test IDs for Login module
ids = generate_test_ids("LOGIN", 5)
```

```
print(ids)
# ['LOGIN_TC_001', 'LOGIN_TC_002', 'LOGIN_TC_003', 'LOGIN_TC_004', 'LOGIN_TC_005']
```

```
# SDET Scenario: Pagination Testing

def test_pagination(total_items, page_size):
    """Calculate pagination test scenarios"""
    pages = []
    for start in range(0, total_items, page_size):
        end = min(start + page_size, total_items)
        pages.append({
            "page_start": start,
            "page_end": end,
            "items_on_page": end - start
        })
    return pages

# Test pagination of 23 items with page size 10
result = test_pagination(23, 10)
for page in result:
    print(page)
# {'page_start': 0, 'page_end': 10, 'items_on_page': 10}
# {'page_start': 10, 'page_end': 20, 'items_on_page': 10}
# {'page_start': 20, 'page_end': 23, 'items_on_page': 3}
```

🎓 Practice Problems

Problem 1: Easy 🟢

Scenario: You need to print numbers from 1 to 10.
Challenge: Write a loop that prints: `1 2 3 4 5 6 7 8 9 10`
Hint: Remember, `range(1, 11)` goes from 1 to 10!

Problem 2: Medium 🟡

Scenario: You're validating a PIN code. You need to count how many digits are in a number.
Challenge: Write a function `count_digits(n)` that returns how many digits are in a positive integer.
Examples: `count_digits(5847)` → 4, `count_digits(100)` → 3

Problem 3: Application 🔴

Scenario: As an SDET, you need to generate test data with IDs counting backward.
Challenge: Create a function that generates test IDs from 100 down to 1, formatted as `"CLEANUP_100"`, `"CLEANUP_099"`, etc.
Bonus: The IDs should be zero-padded to 3 digits.

⚠️ Common Mistakes

❌ **Mistake 1**: `range(10)` includes 10
✅ **Fix**: `range(10)` is 0-9. For 0-10, use `range(11)`

❌ **Mistake 2**: Forgetting the step in backward counting
✅ **Fix**: `range(10, 0)` doesn't work! Use `range(10, 0, -1)`

❌ **Mistake 3**: Off-by-one errors in pagination
✅ **Fix**: Always check: Does page 1 start at index 0 or 1?

❌ **Mistake 4**: Losing digits during extraction
✅ **Fix**: Digits are extracted in reverse order. Remember to reverse at the end!

🔑 Key Takeaways

✅ **Counting forward** = `range(start, stop)` — excludes stop!
✅ **Counting backward** = `range(start, stop, -1)` — needs negative step
✅ **Skip counting** = `range(start, stop, step)` — jump by step
✅ **Extract last digit** = `number % 10`
✅ **Remove last digit** = `number // 10`
✅ **Every loop is counting** — just with different parameters

---

🔗 Related Topics

- **Module 3: Number Properties** → Even/Odd uses the % operator
- **Module 5: Division & Remainder** → Deep dive into % and //
- **Module 50: Modulo Operation** → Advanced uses of remainder

---

📝 Quick Reference Card

What I Want	How To Do It	Code Pattern
Count 0 to 9	Forward by 1	`range(10)`
Count 1 to 10	Forward by 1, start at 1	`range(1, 11)`
Count 10 to 1	Backward by 1	`range(10, 0, -1)`
Even numbers 0-20	Skip by 2	`range(0, 21, 2)`
Get last digit	Remainder of 10	`n % 10`
Remove last digit	Integer divide by 10	`n // 10`
Iterate with index	Enumerate	`for i, val in enumerate(list)`

💾 *Save as:* `Module_01_Number_Fundamentals.md`

Next up: Module 2 - Comparison & Logic! 🚀