[Calculation Name] Specification

[Your Name]

5 February 2026

Table of contents

Field	Value
System/Area	[e.g., Pensions, Finance, HR]
Version	1.0
Author	[Your Name]
Date	[Date]
Status	Draft

0.1 1. Purpose

What does this calculation do? Write 1-2 sentences.

This specification defines how to calculate [describe the calculation].

Business context: [Why is this calculation needed? What process uses it?]

0.2 2. Inputs

What data is needed to perform this calculation?

Input	Description	Example
`principal`	Starting amount	£10,000
`rate`	Annual interest rate	5%
`years`	Number of years	3

```
# ============================================
# INPUT VALUES - Edit these to test scenarios
# ============================================

principal = 10000.00    # Starting amount (£)
rate = 0.05             # Annual rate (5% = 0.05)
years = 3               # Number of years
```

0.3 3. Parameters

What rules or constants apply? These typically don't change between calculations.

```
# ============================================
# PARAMETERS - Usually don't change these
# ============================================

# Example: compound frequency
compounds_per_year = 1  # 1 = annual, 12 = monthly

# Example: minimum/maximum limits
minimum_result = 0
maximum_result = 1000000
```

0.4 4. Calculation

Break down the calculation into clear steps. Each step should have: 1. *A markdown cell explaining what we're doing* 2. *A code cell that does it and shows the result*

0.4.1 Step 1: [Name of first step]

Explain the formula or logic:

Formula: `result = principal × (1 + rate)^years`

```
# Step 1: Calculate compound growth
growth_factor = (1 + rate) ** years
result_step1 = principal * growth_factor

print(f"Principal:      £{principal:,.2f}")
print(f"Rate:           {rate:.2%}")
print(f"Years:          {years}")
print(f"Growth factor:  {growth_factor:.4f}")
print(f"Result:         £{result_step1:,.2f}")
```

```
Principal:      £10,000.00
Rate:           5.00%
Years:          3
```

```
Growth factor:  1.1576
Result:         £11,576.25
```

0.4.2 Step 2: [Name of second step]

Add more steps as needed. Delete this section if not required.

```
# Step 2: Calculate interest earned
interest_earned = result_step1 - principal

print(f"Interest earned: £{interest_earned:,.2f}")
```

```
Interest earned: £1,576.25
```

0.4.3 Step 3: Apply any conditions or limits

Example of conditional logic — delete if not needed.

```
# Step 3: Apply limits (example)
if result_step1 < minimum_result:
    final_result = minimum_result
    print(f"Result below minimum, using: £{minimum_result:,.2f}")
elif result_step1 > maximum_result:
    final_result = maximum_result
    print(f"Result above maximum, capping at: £{maximum_result:,.2f}")
else:
    final_result = result_step1
    print(f"Result within limits: £{final_result:,.2f}")
```

```
Result within limits: £11,576.25
```

0.5 5. Result

```
print("="*45)
print(f"FINAL RESULT: £{final_result:,.2f}")
print("="*45)
```

```
=============================================
FINAL RESULT: £11,576.25
=============================================
```

0.6 6. Summary

A complete summary showing inputs, calculation, and output.

```
print("CALCULATION SUMMARY")
print("="*45)
print(f"")
print(f"INPUTS:")
print(f"  Principal:           £{principal:>12,.2f}")
print(f"  Annual rate:          {rate:>12.2%}")
print(f"  Years:                {years:>12}")
print(f"")
print(f"CALCULATION:")
```

```
print(f"  Growth factor:         {growth_factor:>12.4f}")
print(f"  Interest earned:     £{interest_earned:>12,.2f}")
print(f"")
print(f"RESULT:")
print(f"  Final amount:        £{final_result:>12,.2f}")
print(f"")
print("="*45)
```

```
CALCULATION SUMMARY
=============================================

INPUTS:
  Principal:           £   10,000.00
  Annual rate:                 5.00%
  Years:                           3

CALCULATION:
  Growth factor:             1.1576
  Interest earned:     £    1,576.25

RESULT:
  Final amount:        £   11,576.25

=============================================
```

0.7 7. Test Cases

Document test scenarios to verify the calculation works correctly.

Scenario	Principal	Rate	Years	Expected Result
Basic case	£10,000	5%	3	£11,576.25
Zero years	£10,000	5%	0	£10,000.00
Zero rate	£10,000	0%	5	£10,000.00
High rate	£10,000	10%	10	£25,937.42

0.8 8. Edge Cases & Business Rules

How should special situations be handled?

Scenario	Rule
Negative principal	Not allowed — reject input
Rate above 100%	Allow (valid in some contexts)
Fractional years	Round to 2 decimal places

0.9 9. Sign-Off

Role	Name	Date	Signature
Author			
Reviewer			
Approver			

1 Appendix: Building Blocks Reference

Delete this appendix from your final spec — it's here to help you while writing.

1.1 A1. Common Calculation Patterns

Copy and adapt these for your spec.

```
# --- COMPOUND GROWTH / REVALUATION ---
# Use for: interest, inflation, pension revaluation

original_value = 1000
annual_rate = 0.03      # 3%
num_years = 5

future_value = original_value * (1 + annual_rate) ** num_years
print(f"Compound growth: £{original_value:,.2f} → £{future_value:,.2f}")
```

```
Compound growth: £1,000.00 → £1,159.27
```

```
# --- SIMPLE INTEREST ---
# Use for: simple interest calculations

principal = 1000
rate = 0.05
time = 3

interest = principal * rate * time
total = principal + interest
print(f"Simple interest: £{interest:,.2f}, Total: £{total:,.2f}")
```

```
Simple interest: £150.00, Total: £1,150.00
```

```
# --- PERCENTAGE OF A VALUE ---
# Use for: tax, deductions, fractions

gross_amount = 50000
percentage = 0.20       # 20%

deduction = gross_amount * percentage
net_amount = gross_amount - deduction
print(f"Gross: £{gross_amount:,.2f}, Deduction: £{deduction:,.2f}, Net: £{net_amount:,.2f}")
```

```
Gross: £50,000.00, Deduction: £10,000.00, Net: £40,000.00
```

```
# --- PRO-RATA CALCULATION ---
# Use for: part-year, part-time adjustments
```

```python
full_amount = 12000     # Annual amount
months_worked = 7       # Out of 12

pro_rata_amount = full_amount * (months_worked / 12)
print(f"Pro-rata ({months_worked}/12): £{pro_rata_amount:,.2f}")
```

```
Pro-rata (7/12): £7,000.00
```

```python
# --- APPLY A FACTOR FROM A TABLE ---
# Use for: actuarial factors, tax bands, lookup values

factors = {
    "Band A": 1.00,
    "Band B": 0.85,
    "Band C": 0.70,
}

base_value = 1000
band = "Band B"

adjusted_value = base_value * factors[band]
print(f"{band} factor: {factors[band]}, Result: £{adjusted_value:,.2f}")
```

```
Band B factor: 0.85, Result: £850.00
```

```python
# --- TIERED / BANDED CALCULATION ---
# Use for: tax bands, contribution tiers, sliding scales

income = 55000

# Tax bands (simplified UK example)
if income <= 12570:
    tax = 0
elif income <= 50270:
    tax = (income - 12570) * 0.20
elif income <= 125140:
    tax = (50270 - 12570) * 0.20 + (income - 50270) * 0.40
else:
    tax = (50270 - 12570) * 0.20 + (125140 - 50270) * 0.40 + (income - 125140) * 0.45

print(f"Income: £{income:,.2f}, Tax: £{tax:,.2f}")
```

```
Income: £55,000.00, Tax: £9,432.00
```

```python
# --- MINIMUM / MAXIMUM LIMITS ---
# Use for: caps, floors, clamping values

calculated_value = 1500
minimum_allowed = 500
maximum_allowed = 1000

# Method 1: Using if/else
if calculated_value < minimum_allowed:
    final_value = minimum_allowed
elif calculated_value > maximum_allowed:
    final_value = maximum_allowed
```

```
else:
    final_value = calculated_value

# Method 2: Using min/max (more concise)
final_value = max(minimum_allowed, min(calculated_value, maximum_allowed))

print(f"Calculated: £{calculated_value:,.2f}, After limits: £{final_value:,.2f}")
```

```
Calculated: £1,500.00, After limits: £1,000.00
```

```
# --- ROUNDING ---
# Use for: currency amounts, factors

value = 1234.5678

rounded_2dp = round(value, 2)      # 2 decimal places
rounded_whole = round(value, 0)    # Whole number
rounded_down = int(value)          # Always rounds down (truncate)

print(f"Original: {value}")
print(f"2 d.p.:   {rounded_2dp}")
print(f"Whole:    {rounded_whole}")
print(f"Truncate: {rounded_down}")
```

```
Original: 1234.5678
2 d.p.:   1234.57
Whole:    1235.0
Truncate: 1234
```

1.2 A2. Lookup Table Examples

Different ways to structure reference data.

```
# --- SIMPLE KEY-VALUE LOOKUP ---

status_codes = {
    "A": "Active",
    "D": "Deferred",
    "P": "Pensioner",
    "X": "Transferred out",
}

code = "D"
print(f"Code {code} = {status_codes[code]}")
```

```
Code D = Deferred
```

```
# --- NUMERIC LOOKUP (e.g., factors by age) ---

factors_by_age = {
    55: 0.720,
    56: 0.752,
    57: 0.786,
    58: 0.822,
    59: 0.860,
    60: 0.900,
```

```
    61: 0.940,
    62: 0.970,
    63: 0.985,
    64: 0.995,
    65: 1.000,
}

age = 58
factor = factors_by_age[age]
print(f"Factor at age {age}: {factor}")
```

```
Factor at age 58: 0.822
```

```
# --- TWO-DIMENSIONAL LOOKUP (e.g., by age AND category) ---

factors_2d = {
    "Standard": {55: 0.70, 60: 0.85, 65: 1.00},
    "Enhanced": {55: 0.75, 60: 0.90, 65: 1.00},
}

category = "Enhanced"
age = 60
factor = factors_2d[category][age]
print(f"Factor for {category} at age {age}: {factor}")
```

```
Factor for Enhanced at age 60: 0.9
```

```
# --- SAFE LOOKUP WITH DEFAULT ---
# Use .get() to avoid errors if key doesn't exist

factors = {60: 1.0, 65: 1.1, 70: 1.2}

age = 67  # Not in the table!

# This would crash: factors[age]
# This returns a default instead:
factor = factors.get(age, 1.0)  # Returns 1.0 if age not found

print(f"Factor for age {age}: {factor} (default used)")
```

```
Factor for age 67: 1.0 (default used)
```

1.3 A3. Formatting Output

How to display results clearly.

```
# --- NUMBER FORMATTING EXAMPLES ---

amount = 15432.567
rate = 0.0325
factor = 0.826543

print("Currency:")
print(f"  Basic:          £{amount}")           # £15432.567
print(f"  2 decimals:     £{amount:.2f}")       # £15432.57
print(f"  With commas:    £{amount:,.2f}")      # £15,432.57
```

```
print(f"  Right-aligned: £{amount:>12,.2f}")   # £   15,432.57
print()
print("Percentages:")
print(f"  As decimal:    {rate}")              # 0.0325
print(f"  As percent:    {rate:.2%}")          # 3.25%
print()
print("Factors:")
print(f"  4 decimals:    {factor:.4f}")        # 0.8265
print(f"  6 decimals:    {factor:.6f}")        # 0.826543
```

```
Currency:
  Basic:         £15432.567
  2 decimals:    £15432.57
  With commas:   £15,432.57
  Right-aligned: £   15,432.57

Percentages:
  As decimal:    0.0325
  As percent:    3.25%

Factors:
  4 decimals:    0.8265
  6 decimals:    0.826543
```

```
# --- ALIGNED OUTPUT TABLE ---

items = [
    ("Basic pension", 12500.00),
    ("GMP addition", 3200.50),
    ("Bonus", 800.00),
]

print("Item                    Amount")
print("-" * 35)
total = 0
for name, value in items:
    print(f"{name:<20} £{value:>10,.2f}")
    total += value
print("-" * 35)
print(f"{'TOTAL':<20} £{total:>10,.2f}")
```

```
Item                    Amount
-----------------------------------
Basic pension        £ 12,500.00
GMP addition         £  3,200.50
Bonus                £    800.00
-----------------------------------
TOTAL                £ 16,500.50
```

End of template — delete the Appendix section when creating your actual spec.