Module 2: Python Fundamentals for Finance

Lesson 1: Python Basics for Financial Modeling

Introduction

Welcome to Python programming for finance!

In this module, you'll learn Python from scratch - no programming experience required. Unlike generic Python courses, every example here comes from real financial modeling scenarios you'll encounter at PE Club.

What makes this different?

- Every concept explained with finance examples (not abstract programming theory)
- Build financial models from day one
- Learn by doing - you'll write code, not just read it
- Direct Excel-to-Python translations

What you'll learn:

- How to store financial data (numbers, text, lists of cash flows)
- How to perform calculations (like Excel formulas, but more powerful)
- How to build reusable financial functions
- How to create company models that update automatically

Before you start:

- ✅ Complete Module 01 (Python environment set up)
- ✅ Have VS Code open with your `financial-modeling` folder
- ✅ Virtual environment activated (you should see `(venv)` in terminal)

How to use this module:

1. Read each section
2. Type (don't copy-paste!) the code into a new Python file
3. Run it and see the results
4. Experiment - change numbers and see what happens
5. Complete the practice exercises

Let's start coding! 🚀

Setting Up Your First Python File

Before we dive into code, let's create a proper workspace:

1. **Create a new Python file:**
    - In VS Code Explorer (left sidebar), right-click on your `financial-modeling` folder
    - Select "New File"

- Name it: `Module_02_Python_Fundamentals/python_basics.py`
- If the folder doesn't exist, create it first: Right-click → New Folder → `Module_02_Python_Fundamentals`

2. **What you'll see:**
   - A blank file opens in the editor
   - VS Code recognizes it's Python (see "Python" in bottom-right)
   - You might see the Python version displayed
3. **Test your setup:**

   Type this simple code:

```
print("Hello from Python!")
print("Ready to build financial models!")
```

4. **Run it:**
   - Click the ▶ play button (top-right)
   - Terminal opens and shows:

```
Hello from Python!
Ready to build financial models!
```

Congratulations! You just ran your first Python program. Every financial model starts this way - one line of code at a time.

Python Data Types for Finance

What are data types? Think of them as different types of information you'd put in Excel cells:

- Numbers (like revenue: 1,500,000)
- Text (like company name: "Apple Inc.")
- Lists (like years: 2021, 2022, 2023, 2024, 2025)

Python has specific ways to work with each type. Let's learn them through finance examples.

Numbers in Finance

Two types of numbers in Python:

1. **Integers** (whole numbers) - for counting things
2. **Floats** (decimal numbers) - for money, percentages, ratios

Create a new file: `numbers_example.py`

Type this code (read the comments - anything after # is just explanation):

```
# Integers - for counting items, years, periods
shares_outstanding = 1000000  # 1 million shares
projection_years = 5
transaction_year = 2025

print(f"Shares: {shares_outstanding:,}")  # :, adds commas
print(f"Years to project: {projection_years}")

# Floats - for monetary values, rates, multiples
stock_price = 45.67
interest_rate = 0.065  # 6.5% (in decimal form)
ev_ebitda_multiple = 12.5
revenue = 1_500_000_000  # $1.5B (underscores for readability - Python
ignores them)

print(f"Stock price: ${stock_price}")
print(f"Interest rate: {interest_rate:.1%}")  # :.1% formats as percentage

# Financial calculations (just like Excel formulas!)
market_cap = shares_outstanding * stock_price
print(f"\nMarket Cap: ${market_cap:,.2f}")  # :,.2f = comma separator, 2
decimals
```

Run this file (▶ button)

What you should see:

```
Shares: 1,000,000
Years to project: 5
Stock price: $45.67
Interest rate: 6.5%

Market Cap: $45,670,000.00
```

Try this:

- Change `stock_price` to 50.00 and run again
- See how market cap updates automatically? That's the power of code!

Key takeaways:

- `=` assigns a value to a variable (like naming an Excel cell)
- `*` multiplies (just like Excel)
- `print()` displays results (like viewing a cell value)
- `f"..."` creates formatted strings (mix text and numbers)

Strings - Working with Text in Finance

What are strings? Text data - company names, tickers, labels. In Python, text goes in quotes: `"like this"` or `'like this'`.

Create a new file: `strings_example.py`

```
# Strings - for labels and text data
company_name = "TechCo Inc."
ticker = "TECH"
sector = "Technology"

# Print them out
print(company_name)
print(ticker)
print(sector)

# String formatting (f-strings) - combine text and variables
print(f"{company_name} ({ticker}) - {sector}")
# Output: TechCo Inc. (TECH) - Technology

# Creating labels for financial statements (like Excel row headers)
line_items = ["Revenue", "COGS", "Gross Profit", "EBITDA", "Net Income"]

# Print each item
for item in line_items:
    print(f"- {item}")
```

Run it and you'll see:

```
TechCo Inc.
TECH
Technology
TechCo Inc. (TECH) - Technology
- Revenue
- COGS
- Gross Profit
- EBITDA
- Net Income
```

String tricks for finance:

```
# Uppercase/lowercase (useful for cleaning data)
ticker = "aapl"
ticker_clean = ticker.upper()
print(ticker_clean)  # AAPL

# Replace text
company = "Apple Inc."
short_name = company.replace(" Inc.", "")
```

```
print(short_name)  # Apple

# Check if text contains something
if "Inc." in company:
    print("This is an incorporated company")
```

Lists - Your First Financial Time Series

What are lists? Collections of items in order - perfect for years, revenues over time, cash flows. Like an Excel column!

Create a new file: `lists_example.py`

```
# Lists - for time series data (like Excel columns)
# Revenue projections over 5 years (in millions)
revenues = [100, 115, 132, 152, 175]

# Years
years = [2025, 2026, 2027, 2028, 2029]

print("Revenue Projections:")
print(revenues)
print("\nYears:")
print(years)

# Accessing specific elements (Python counts from 0!)
first_year_revenue = revenues[0]  # First item: 100
last_year_revenue = revenues[-1]   # Last item: 175
second_year = years[1]             # Second item: 2026

print(f"\nFirst year revenue: ${first_year_revenue}M")
print(f"Last year revenue: ${last_year_revenue}M")
print(f"Second year: {second_year}")

# List operations (like Excel functions)
total_revenue = sum(revenues)
avg_revenue = sum(revenues) / len(revenues)
max_revenue = max(revenues)
min_revenue = min(revenues)

print(f"\nTotal revenue (5 years): ${total_revenue}M")
print(f"Average revenue: ${avg_revenue:.1f}M")
print(f"Highest year: ${max_revenue}M")
print(f"Lowest year: ${min_revenue}M")

# Adding to lists
revenues.append(200)  # Add 2030 projection
print(f"\nAfter adding 2030: {revenues}")

# Combining lists (like matching Excel columns)
```

```
print("\nYear-by-year breakdown:")
for year, revenue in zip(years, revenues):
    print(f"{year}: ${revenue}M")
```

What you should see:

```
Revenue Projections:
[100, 115, 132, 152, 175]

Years:
[2025, 2026, 2027, 2028, 2029]

First year revenue: $100M
Last year revenue: $175M
Second year: 2026

Total revenue (5 years): $574M
Average revenue: 114.8M
Highest year: $175M
Lowest year: $100M

After adding 2030: [100, 115, 132, 152, 175, 200]

Year-by-year breakdown:
2025: $100M
2026: $115M
2027: $132M
2028: $152M
2029: $175M
```

Try this:

- Add more years to the revenue list
- Calculate the growth from first to last year
- Find which year had the highest revenue

Dictionaries - Financial Data Structures

What are dictionaries? Data with labels (key-value pairs). Like a mini-database or Excel table with named columns. Perfect for company financials!

Create a new file: `dictionaries_example.py`

```
# Dictionary - financial data with labels
# Think of it like a single company's data row in Excel
financials = {
    "revenue": 1500,
    "ebitda": 300,
```

```
    "net_income": 150,
    "total_debt": 400,
    "cash": 100
}

# Access values by their labels
print(f"Revenue: ${financials['revenue']}M")
print(f"EBITDA: ${financials['ebitda']}M")

# Calculate new metrics (like Excel formulas)
net_debt = financials["total_debt"] - financials["cash"]
ebitda_margin = financials["ebitda"] / financials["revenue"]

print(f"\nNet Debt: ${net_debt}M")
print(f"EBITDA Margin: {ebitda_margin:.1%}")

# Add new items
financials["capex"] = 50
print(f"\nCapEx: ${financials['capex']}M")

# See all the data
print("\nAll Financial Data:")
for metric, value in financials.items():
    print(f"  {metric}: ${value}M")

# Multiple companies (list of dictionaries)
companies = [
    {"name": "TechCo", "revenue": 1500, "ebitda": 300},
    {"name": "RetailCo", "revenue": 2000, "ebitda": 250},
    {"name": "BankCo", "revenue": 800, "ebitda": 200}
]

print("\nComp Table:")
print(f"{'Company':<15} {'Revenue':<12} {'EBITDA':<12} {'Margin'}")
print("-" * 50)

for company in companies:
    margin = company["ebitda"] / company["revenue"]
    print(f"{company['name']:<15} ${company['revenue']:<11,} "
          f"${company['ebitda']:<11,} {margin:.1%}")
```

What you should see:

```
Revenue: $1500M
EBITDA: $300M

Net Debt: $300M
EBITDA Margin: 20.0%

CapEx: $50M

All Financial Data:
```

```
  revenue: $1500M
  ebitda: $300M
  net_income: $150M
  total_debt: $400M
  cash: $100M
  capex: $50M

Comp Table:
Company         Revenue      EBITDA       Margin
--------------------------------------------------
TechCo          $1,500       $300         20.0%
RetailCo        $2,000       $250         12.5%
BankCo          $800         $200         25.0%
```

This is powerful! You just created a comparable companies table in Python. No Excel needed!

Functions - Build Your Own Financial Formulas

What are functions? Reusable code blocks - like creating custom Excel functions, but way more powerful. Write once, use forever!

Why use functions?

- Avoid repeating code
- Make complex calculations simple to use
- Easy to test and debug
- Professional standard in finance

Create a new file: `functions_example.py`

```python
# Functions - like custom Excel formulas

# Simple function
def calculate_margin(revenue, cost):
    """Calculate profit margin"""
    margin = (revenue - cost) / revenue
    return margin

# Use the function
revenue = 1000
cogs = 600
gross_margin = calculate_margin(revenue, cogs)

print(f"Revenue: ${revenue}M")
print(f"COGS: ${cogs}M")
print(f"Gross Margin: {gross_margin:.1%}")

# More complex: WACC calculation
def calculate_wacc(equity_value, debt_value, cost_of_equity, cost_of_debt,
tax_rate):
```

```
    """
    Calculate Weighted Average Cost of Capital (WACC)

    Formula: WACC = (E/V × Re) + (D/V × Rd × (1-Tc))
    This is the blended cost of capital used in DCF models.

    Parameters:
    -----------
    equity_value : float
        Market value of equity (in millions)
    debt_value : float
        Market value of debt (in millions)
    cost_of_equity : float
        Required return on equity (e.g., 0.12 for 12%)
    cost_of_debt : float
        Interest rate on debt (e.g., 0.05 for 5%)
    tax_rate : float
        Corporate tax rate (e.g., 0.25 for 25%)

    Returns:
    --------
    float
        WACC as decimal (e.g., 0.095 for 9.5%)
    """
    total_value = equity_value + debt_value
    equity_weight = equity_value / total_value
    debt_weight = debt_value / total_value

    wacc = (equity_weight * cost_of_equity) + \
           (debt_weight * cost_of_debt * (1 - tax_rate))

    return wacc

# Example company
equity = 700  # Market cap
debt = 300    # Total debt
coe = 0.12    # 12% cost of equity
cod = 0.05    # 5% cost of debt
tax = 0.25    # 25% tax rate

wacc = calculate_wacc(equity, debt, coe, cod, tax)

print(f"\n--- WACC Calculation ---")
print(f"Equity Value: ${equity}M")
print(f"Debt Value: ${debt}M")
print(f"Cost of Equity: {coe:.1%}")
print(f"Cost of Debt: {cod:.1%}")
print(f"Tax Rate: {tax:.1%}")
print(f"\nWACC: {wacc:.2%}")

# DCF Present Value function
def present_value(cash_flow, discount_rate, year):
    """
    Calculate present value of a future cash flow.
```

```
    Formula: PV = CF / (1 + r)^n

    Parameters:
    -----------
    cash_flow : float
        Future cash flow amount
    discount_rate : float
        Discount rate (WACC)
    year : int
        Number of years in the future

    Returns:
    --------
    float
        Present value of the cash flow
    """
    pv = cash_flow / (1 + discount_rate) ** year
    return pv

# Project multiple cash flows
cash_flows = [100, 110, 121, 133, 146]
discount_rate = 0.10
years = [1, 2, 3, 4, 5]

print("\n--- DCF Calculation ---")
print(f"Discount Rate: {discount_rate:.1%}\n")
print(f"{'Year':<6} {'Cash Flow':<12} {'Present Value':<15}")
print("-" * 35)

total_pv = 0
for year, cf in zip(years, cash_flows):
    pv = present_value(cf, discount_rate, year)
    total_pv += pv
    print(f"{year:<6} ${cf:<11,.0f} ${pv:<14,.2f}")

print("-" * 35)
print(f"{'Total':<6} ${sum(cash_flows):<11,.0f} ${total_pv:<14,.2f}")
```

What you should see:

```
Revenue: $1000M
COGS: $600M
Gross Margin: 40.0%

--- WACC Calculation ---
Equity Value: $700M
Debt Value: $300M
Cost of Equity: 12.0%
Cost of Debt: 5.0%
Tax Rate: 25.0%
```

```
WACC: 9.53%

--- DCF Calculation ---
Discount Rate: 10.0%

Year   Cash Flow    Present Value
-----------------------------------
1      $100         $90.91
2      $110         $90.91
3      $121         $90.92
4      $133         $90.88
5      $146         $90.66
-----------------------------------
Total  $610         $454.28
```

Try this:

- Change the WACC inputs and see how it affects the result
- Add more years to the DCF calculation
- Create your own function to calculate P/E ratio

Functions = Excel formulas on steroids! 💪

Loops - Automate Repetitive Tasks

What are loops? Run the same code multiple times automatically. Like copy-pasting Excel formulas down a column, but automatic!

Two types you'll use:

Create a new file: `loops_example.py`

```python
# FOR LOOPS - when you know how many times to repeat

# Example 1: Revenue projections
base_revenue = 100
growth_rate = 0.15  # 15% annual growth
years = 5

print("Revenue Projections (15% growth):\n")
print(f"{'Year':<6} {'Revenue ($M)':<15}")
print("-" * 25)

revenue = base_revenue
for year in range(1, years + 1):
    print(f"{year:<6} ${revenue:<14,.2f}")
    revenue = revenue * (1 + growth_rate)  # Apply growth

print("\n" + "=" * 40 + "\n")
```

```
# Example 2: Building an income statement
print("Income Statement Builder:\n")

revenues = [1000, 1150, 1320, 1520, 1750]
cogs_margin = 0.60  # COGS is 60% of revenue
opex_margin = 0.20  # OpEx is 20% of revenue

print(f"{'Year':<6} {'Revenue':<10} {'COGS':<10} {'OpEx':<10} {'EBITDA':
<10}")
print("-" * 50)

for i, revenue in enumerate(revenues, start=1):
    cogs = revenue * cogs_margin
    opex = revenue * opex_margin
    ebitda = revenue - cogs - opex

    print(f"{i:<6} ${revenue:<9,.0f} ${cogs:<9,.0f} "
          f"${opex:<9,.0f} ${ebitda:<9,.0f}")

print("\n" + "=" * 40 + "\n")

# WHILE LOOPS - repeat until a condition is met
# Example: How long until debt is paid off?

print("Debt Paydown Schedule:\n")

initial_debt = 500  # $500M
annual_payment = 75  # $75M per year
interest_rate = 0.05  # 5% interest

debt = initial_debt
year = 0

print(f"{'Year':<6} {'Starting Debt':<15} {'Interest':<12} {'Payment':<12}
{'Ending Debt':<15}")
print("-" * 60)

while debt > 0:
    year += 1
    interest = debt * interest_rate
    payment = min(annual_payment, debt + interest)  # Don't overpay
    ending_debt = debt + interest - payment

    print(f"{year:<6} ${debt:<14,.2f} ${interest:<11,.2f} "
          f"${payment:<11,.2f} ${ending_debt:<14,.2f}")

    debt = ending_debt

    if year > 20:  # Safety limit
        print("\nWARNING: Debt not paid off in 20 years!")
        break

print(f"\nDebt fully paid in {year} years! 🎉")
```

What you should see:

```
Revenue Projections (15% growth):

Year   Revenue ($M)
------------------------
1      $100.00
2      $115.00
3      $132.25
4      $152.09
5      $174.90

======================================

Income Statement Builder:

Year   Revenue    COGS       OpEx       EBITDA
-----------------------------------------------
1      $1,000     $600       $200       $200
2      $1,150     $690       $230       $230
3      $1,320     $792       $264       $264
4      $1,520     $912       $304       $304
5      $1,750     $1,050     $350       $350

======================================

Debt Paydown Schedule:

Year   Starting Debt   Interest     Payment      Ending Debt
-------------------------------------------------------------
1      $500.00         $25.00       $75.00       $450.00
2      $450.00         $22.50       $75.00       $397.50
3      $397.50         $19.88       $75.00       $342.38
4      $342.38         $17.12       $75.00       $284.49
5      $284.49         $14.22       $75.00       $223.72
6      $223.72         $11.19       $75.00       $159.90
7      $159.90         $8.00        $75.00       $92.90
8      $92.90          $4.65        $75.00       $22.54
9      $22.54          $1.13        $23.67       $0.00

Debt fully paid in 9 years! 🎉
```

This just replaced 3 complex Excel models! Notice how the code:

- Projects revenue automatically
- Builds income statements for multiple years
- Calculates debt paydown with interest

Try this:

- Change the growth rate to 20%

- Modify the debt payment amount
- Add a tax line to the income statement

Conditionals - Making Decisions in Code

What are conditionals? Make different decisions based on conditions - like Excel IF statements, but more powerful!

Create a new file: `conditionals_example.py`

```
# CONDITIONALS - Making decisions based on data

# Example 1: Credit Rating Classification
def classify_credit_rating(interest_coverage):
    """
    Classify credit rating based on interest coverage ratio.

    Interest Coverage = EBIT / Interest Expense

    This is used in debt analysis to assess creditworthiness.

    Parameters:
    -----------
    interest_coverage : float
        Interest coverage ratio

    Returns:
    --------
    str
        Credit rating classification
    """
    if interest_coverage > 8:
        return "AAA - Excellent"
    elif interest_coverage > 6:
        return "AA - Very Good"
    elif interest_coverage > 4:
        return "A - Good"
    elif interest_coverage > 2.5:
        return "BBB - Adequate (Investment Grade)"
    elif interest_coverage > 1.5:
        return "BB - Speculative"
    elif interest_coverage > 1.0:
        return "B - Highly Speculative"
    else:
        return "CCC or below - High Risk"

# Test with different companies
companies = [
    {"name": "Safe Corp", "ebit": 500, "interest": 50},
    {"name": "Levered Co", "ebit": 200, "interest": 80},
    {"name": "Risky Inc", "ebit": 100, "interest": 120}
```

```
]

print("Credit Rating Analysis:\n")
print(f"{'Company':<15} {'EBIT':<10} {'Interest':<10} {'Coverage':<10}
{'Rating'}")
print("-" * 70)

for company in companies:
    coverage = company["ebit"] / company["interest"]
    rating = classify_credit_rating(coverage)

    print(f"{company['name']:<15} ${company['ebit']:<9,.0f} "
          f"${company['interest']:<9,.0f} {coverage:<10.2f}x {rating}")

print("\n" + "=" * 70 + "\n")

# Example 2: Investment Decision Logic
def investment_decision(irr, moic, hold_period_years):
    """
    Make PE investment decision based on return thresholds.

    PE Club's typical hurdle rates:
    - IRR: Minimum 20%
    - MOIC: Minimum 2.5x
    - Hold period: Maximum 7 years

    Parameters:
    -----------
    irr : float
        Internal Rate of Return (e.g., 0.25 for 25%)
    moic : float
        Multiple on Invested Capital (e.g., 3.0 for 3.0x)
    hold_period_years : int
        Investment holding period

    Returns:
    --------
    str
        Investment recommendation
    """
    # Check all criteria
    meets_irr = irr >= 0.20
    meets_moic = moic >= 2.5
    meets_period = hold_period_years <= 7

    if meets_irr and meets_moic and meets_period:
        if irr >= 0.30 and moic >= 3.5:
            return "STRONG BUY - Exceptional returns"
        elif irr >= 0.25 and moic >= 3.0:
            return "BUY - Above target returns"
        else:
            return "BUY - Meets minimum thresholds"
    elif not meets_period:
        return "PASS - Hold period too long"
```

```
    elif not meets_irr:
        return f"PASS - IRR below 20% hurdle (actual: {irr:.1%})"
    elif not meets_moic:
        return f"PASS - MOIC below 2.5x hurdle (actual: {moic:.1f}x)"
    else:
        return "PASS - Does not meet investment criteria"

# Evaluate multiple deals
deals = [
    {"name": "TechCo Buyout", "irr": 0.32, "moic": 3.8, "years": 5},
    {"name": "RetailCo LBO", "irr": 0.22, "moic": 2.6, "years": 6},
    {"name": "ManufactCo", "irr": 0.15, "moic": 2.2, "years": 5},
    {"name": "ServiceCo", "irr": 0.28, "moic": 3.2, "years": 8}
]

print("PE Investment Committee Recommendations:\n")
print(f"{'Deal':<20} {'IRR':<8} {'MOIC':<8} {'Years':<7} {'Decision'}")
print("-" * 80)

for deal in deals:
    decision = investment_decision(deal["irr"], deal["moic"],
deal["years"])

    print(f"{deal['name']:<20} {deal['irr']:<7.1%} {deal['moic']:<7.1f}x "
          f"{deal['years']:<7} {decision}")

print("\n" + "=" * 70 + "\n")

# Example 3: Valuation Multiple Comparison
def compare_to_market(company_multiple, industry_median,
industry_top_quartile):
    """
    Compare company valuation multiple to industry benchmarks.

    Parameters:
    -----------
    company_multiple : float
        Company's trading multiple (e.g., EV/EBITDA)
    industry_median : float
        Industry median multiple
    industry_top_quartile : float
        Industry top quartile multiple

    Returns:
    --------
    str
        Valuation assessment
    """
    discount_to_median = (industry_median - company_multiple) /
industry_median

    if company_multiple >= industry_top_quartile:
        return f"Premium valuation (+{discount_to_median:.1%} vs median)"
    elif company_multiple >= industry_median:
```

```
        return f"Above median (+{discount_to_median:.1%} vs median)"
    elif discount_to_median <= 0.15:
        return f"Slight discount ({discount_to_median:.1%} vs median)"
    elif discount_to_median <= 0.30:
        return f"Moderate discount ({discount_to_median:.1%} vs median) -
INTERESTING"
    else:
        return f"Deep discount ({discount_to_median:.1%} vs median) - DEEP
VALUE?"

# Analyze target company
target_ev_ebitda = 6.5
industry_median = 8.5
industry_top_q = 10.0

assessment = compare_to_market(target_ev_ebitda, industry_median,
industry_top_q)

print("Valuation Analysis:")
print(f"Target Company EV/EBITDA: {target_ev_ebitda:.1f}x")
print(f"Industry Median: {industry_median:.1f}x")
print(f"Industry Top Quartile: {industry_top_q:.1f}x")
print(f"\nAssessment: {assessment}")
```

What you should see:

```
Credit Rating Analysis:

Company         EBIT       Interest   Coverage   Rating
------------------------------------------------------------------------
Safe Corp       $500       $50        10.00x     AAA - Excellent
Levered Co      $200       $80        2.50x      BBB - Adequate
(Investment Grade)
Risky Inc       $100       $120       0.83x      CCC or below - High Risk

========================================================================

PE Investment Committee Recommendations:

Deal                IRR      MOIC     Years   Decision
------------------------------------------------------------------------
------
TechCo Buyout       32.0%    3.8x     5       STRONG BUY - Exceptional
returns
RetailCo LBO        22.0%    2.6x     6       BUY - Meets minimum
thresholds
ManufactCo          15.0%    2.2x     5       PASS - IRR below 20% hurdle
(actual: 15.0%)
ServiceCo           28.0%    3.2x     8       PASS - Hold period too long

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

```
Valuation Analysis:
Target Company EV/EBITDA: 6.5x
Industry Median: 8.5x
Industry Top Quartile: 10.0x

Assessment: Moderate discount (23.5% vs median) - INTERESTING
```

This is how PE analysts think! The code captures real investment decision logic.

Try this:

- Change the IRR hurdle rate to 25%
- Add a new criterion (e.g., minimum revenue)
- Create a function to classify companies by size (small/mid/large cap)

Classes - Building Reusable Financial Models

What are classes? Templates for creating objects that combine data and functions. Think of it like creating your own custom Excel template that calculates everything automatically!

Why use classes in finance?

- Group related data together (company financials)
- Encapsulate calculations (metrics auto-update)
- Reusable across multiple companies/deals
- Professional code organization

Create a new file: `classes_example.py`

```
class Company:
    """
    Financial model for a company - like a reusable Excel template.

    This class stores company financials and automatically calculates
    key metrics. Perfect for building comps tables or screening targets.
    """

    def __init__(self, name, ticker, revenue, ebitda, net_income,
                 total_debt=0, cash=0, shares_outstanding=100,
share_price=10):
        """
        Initialize a company with its financials.

        Parameters:
        -----------
        name : str
            Company name
        ticker : str
            Stock ticker symbol
        revenue : float
```

```
            Annual revenue (millions)
        ebitda : float
            EBITDA (millions)
        net_income : float
            Net income (millions)
        total_debt : float, optional
            Total debt (millions)
        cash : float, optional
            Cash and equivalents (millions)
        shares_outstanding : float, optional
            Shares outstanding (millions)
        share_price : float, optional
            Current share price
        """
        # Store the data
        self.name = name
        self.ticker = ticker
        self.revenue = revenue
        self.ebitda = ebitda
        self.net_income = net_income
        self.total_debt = total_debt
        self.cash = cash
        self.shares_outstanding = shares_outstanding
        self.share_price = share_price

    def market_cap(self):
        """Calculate market capitalization"""
        return self.shares_outstanding * self.share_price

    def enterprise_value(self):
        """Calculate enterprise value: Market Cap + Debt - Cash"""
        return self.market_cap() + self.total_debt - self.cash

    def ebitda_margin(self):
        """Calculate EBITDA margin as percentage"""
        if self.revenue == 0:
            return 0
        return self.ebitda / self.revenue

    def net_margin(self):
        """Calculate net margin as percentage"""
        if self.revenue == 0:
            return 0
        return self.net_income / self.revenue

    def ev_to_ebitda(self):
        """Calculate EV/EBITDA multiple"""
        if self.ebitda == 0:
            return None
        return self.enterprise_value() / self.ebitda

    def pe_ratio(self):
        """Calculate P/E ratio"""
        if self.net_income <= 0:
```

```
            return None
        return self.market_cap() / self.net_income

    def net_debt(self):
        """Calculate net debt"""
        return self.total_debt - self.cash

    def display_summary(self):
        """Print a formatted summary of the company"""
        print(f"\n{'='*60}")
        print(f"{self.name} ({self.ticker})")
        print(f"{'='*60}")
        print(f"\nFinancials (in millions):")
        print(f"  Revenue:              ${self.revenue:>12,.0f}")
        print(f"  EBITDA:               ${self.ebitda:>12,.0f}")
        print(f"  Net Income:           ${self.net_income:>12,.0f}")
        print(f"  Total Debt:           ${self.total_debt:>12,.0f}")
        print(f"  Cash:                 ${self.cash:>12,.0f}")
        print(f"  Net Debt:             ${self.net_debt():>12,.0f}")

        print(f"\nValuation:")
        print(f"  Share Price:          ${self.share_price:>12,.2f}")
        print(f"  Shares Outstanding:
{self.shares_outstanding:>12,.1f}M")
        print(f"  Market Cap:           ${self.market_cap():>12,.0f}M")
        print(f"  Enterprise Value:
${self.enterprise_value():>12,.0f}M")

        print(f"\nMetrics:")
        print(f"  EBITDA Margin:        {self.ebitda_margin():>13.1%}")
        print(f"  Net Margin:           {self.net_margin():>13.1%}")

        ev_ebitda = self.ev_to_ebitda()
        if ev_ebitda:
            print(f"  EV/EBITDA:            {ev_ebitda:>13.1f}x")

        pe = self.pe_ratio()
        if pe:
            print(f"  P/E Ratio:            {pe:>13.1f}x")

        print(f"{'='*60}\n")

# Create company instances
techco = Company(
    name="TechCo Inc.",
    ticker="TECH",
    revenue=1500,
    ebitda=300,
    net_income=150,
    total_debt=400,
    cash=100,
    shares_outstanding=100,
    share_price=15
)
```

```
retailco = Company(
    name="RetailCo Corp.",
    ticker="RETL",
    revenue=2000,
    ebitda=250,
    net_income=120,
    total_debt=600,
    cash=50,
    shares_outstanding=150,
    share_price=12
)

# Display summaries
techco.display_summary()
retailco.display_summary()

# Build a comps table
print("\nComparable Companies Analysis:\n")
print(f"{'Company':<20} {'EV ($M)':<12} {'EBITDA ($M)':<12} {'EV/EBITDA':
<12} {'EBITDA Margin'}")
print("-" * 70)

companies = [techco, retailco]

for company in companies:
    print(f"{company.name:<20} ${company.enterprise_value():<11,.0f} "
          f"${company.ebitda:<11,.0f} {company.ev_to_ebitda():<11.1f}x "
          f"{company.ebitda_margin():<.1%}")

# Calculate industry averages
avg_ev_ebitda = sum(c.ev_to_ebitda() for c in companies) / len(companies)
avg_ebitda_margin = sum(c.ebitda_margin() for c in companies) /
len(companies)

print("-" * 70)
print(f"{'Industry Average':<20} {'':<12} {'':<12} {avg_ev_ebitda:<11.1f}x
{avg_ebitda_margin:<.1%}")
```

What you should see:

```
============================================================
TechCo Inc. (TECH)
============================================================

Financials (in millions):
  Revenue:                    $1,500
  EBITDA:                       $300
  Net Income:                   $150
  Total Debt:                   $400
  Cash:                         $100
  Net Debt:                     $300
```

```
Valuation:
  Share Price:                  $15.00
  Shares Outstanding:            100.0M
  Market Cap:                  $1,500M
  Enterprise Value:            $1,800M

Metrics:
  EBITDA Margin:                 20.0%
  Net Margin:                    10.0%
  EV/EBITDA:                      6.0x
  P/E Ratio:                     10.0x
============================================================

[RetailCo output similar...]

Comparable Companies Analysis:

Company              EV ($M)      EBITDA ($M)  EV/EBITDA    EBITDA Margin
------------------------------------------------------------------------
TechCo Inc.          $1,800       $300         6.0x         20.0%
RetailCo Corp.       $2,350       $250         9.4x         12.5%
------------------------------------------------------------------------
Industry Average                                7.7x         16.2%
```

This is powerful! You just created a reusable company model. Now you can:

- Analyze any company instantly
- Build comp tables automatically
- Update one number and all metrics recalculate

Try this:

- Add a method to calculate ROE (Return on Equity)
- Create a method to project revenue growth
- Add a comparison method to compare two companies

Practice Exercises

Now it's your turn! These exercises will solidify your Python fundamentals using real PE/IB scenarios.

Exercise 1: Revenue Projection Function ⭐

Scenario: You're building a revenue model for a target company. Create a function that projects revenue with different growth rates each year.

```python
def project_revenue(base_revenue, growth_rates):
    """
    Project revenue over multiple years with varying growth rates.
```

```
    Parameters:
    -----------
    base_revenue : float
        Starting revenue (Year 0)
    growth_rates : list
        Annual growth rate for each year (e.g., [0.15, 0.12, 0.10])

    Returns:
    --------
    list
        Revenue projections for each year

    Example:
    --------
    >>> project_revenue(100, [0.15, 0.12, 0.10])
    [115.0, 128.8, 141.68]
    """
    # Your code here
    pass

# Test your function
base = 100
growth = [0.15, 0.12, 0.10, 0.08, 0.06]
revenues = project_revenue(base, growth)

print("Revenue Projections:")
for year, rev in enumerate(revenues, 1):
    print(f"Year {year}: ${rev:.2f}M")
```

Expected output:

```
Revenue Projections:
Year 1: $115.00M
Year 2: $128.80M
Year 3: $141.68M
Year 4: $153.01M
Year 5: $162.19M
```

Hint: Use a loop to apply each growth rate sequentially.

Exercise 2: LBO Returns Calculator ⭐⭐

Scenario: PE Club is evaluating a potential buyout. Calculate the MOIC (Multiple on Invested Capital) and approximate IRR.

```
def calculate_lbo_returns(entry_ev, exit_ev, entry_debt, exit_debt,
equity_invested):
    """
```

```
    Calculate LBO returns (MOIC and approximate IRR).

    In an LBO:
    - Entry: Equity Invested = Entry EV - Entry Debt
    - Exit: Equity Value = Exit EV - Exit Debt
    - MOIC = Exit Equity / Entry Equity
    - Approximate IRR = (MOIC)^(1/years) - 1

    Parameters:
    -----------
    entry_ev : float
        Entry enterprise value
    exit_ev : float
        Exit enterprise value
    entry_debt : float
        Debt at entry
    exit_debt : float
        Debt at exit (usually lower due to paydown)
    equity_invested : float
        Equity check written at entry

    Returns:
    --------
    dict
        Dictionary with 'moic' and 'irr_approx' (assuming 5-year hold)

    Example:
    --------
    >>> calculate_lbo_returns(500, 750, 300, 150, 200)
    {'moic': 3.0, 'irr_approx': 0.246}  # 3.0x, 24.6% IRR
    """
    # Your code here
    pass

# Test case: PE Club's target deal
entry_enterprise_value = 500  # $500M entry valuation
exit_enterprise_value = 750   # $750M exit valuation (5 years later)
debt_at_entry = 300           # $300M debt
debt_at_exit = 150            # $150M debt (paid down)
equity_check = 200            # $200M equity invested

returns = calculate_lbo_returns(
    entry_enterprise_value,
    exit_enterprise_value,
    debt_at_entry,
    debt_at_exit,
    equity_check
)

print(f"LBO Returns Analysis:")
print(f"MOIC: {returns['moic']:.2f}x")
print(f"IRR (5-year hold): {returns['irr_approx']:.1%}")
```

Expected output:

```
LBO Returns Analysis:
MOIC: 3.00x
IRR (5-year hold): 24.6%
```

Hint:

- Exit equity = Exit EV - Exit Debt
- MOIC = Exit equity / Equity invested
- IRR ≈ (MOIC)^(1/5) - 1

Exercise 3: Simple DCF Model ⭐⭐⭐

Scenario: Build a basic DCF calculator that values a company based on projected free cash flows.

```
def simple_dcf(free_cash_flows, wacc, terminal_growth_rate):
    """
    Calculate enterprise value using Discounted Cash Flow.

    Steps:
    1. Discount each cash flow to present value
    2. Calculate terminal value = Final FCF × (1 + g) / (WACC - g)
    3. Discount terminal value to present
    4. Sum all present values

    Parameters:
    -----------
    free_cash_flows : list
        Projected free cash flows for 5 years
    wacc : float
        Weighted average cost of capital (discount rate)
    terminal_growth_rate : float
        Perpetual growth rate (typically 2-3%)

    Returns:
    --------
    dict
        Dictionary with 'pv_cash_flows', 'terminal_value',
'pv_terminal_value', 'enterprise_value'

    Example:
    --------
    >>> simple_dcf([100, 110, 120, 130, 140], 0.10, 0.025)
    {'enterprise_value': 1450.23, ...}
    """
    # Your code here
    pass
```

```
# Test with a target company
fcf_projections = [100, 110, 121, 133, 146]  # Growing cash flows
discount_rate = 0.10  # 10% WACC
terminal_growth = 0.025  # 2.5% perpetuity growth

dcf_result = simple_dcf(fcf_projections, discount_rate, terminal_growth)

print("DCF Valuation:")
print(f"PV of Cash Flows: ${dcf_result['pv_cash_flows']:.2f}M")
print(f"Terminal Value: ${dcf_result['terminal_value']:.2f}M")
print(f"PV of Terminal Value: ${dcf_result['pv_terminal_value']:.2f}M")
print(f"\nEnterprise Value: ${dcf_result['enterprise_value']:.2f}M")
```

Expected output:

```
DCF Valuation:
PV of Cash Flows: $454.28M
Terminal Value: $1,995.00M
PV of Terminal Value: $1,238.76M

Enterprise Value: $1,693.04M
```

Hint:

- PV = CF / (1 + WACC)^year
- Terminal Value = Final FCF × (1 + g) / (WACC - g)
- Remember to discount terminal value back to present!

Exercise 4: Comparable Companies Table ⭐⭐

Scenario: Build a function that creates a comps table from a list of company dictionaries.

```
def create_comps_table(companies):
    """
    Create a comparable companies analysis table.

    Calculate for each company:
    - EV/EBITDA multiple
    - EV/Revenue multiple
    - EBITDA margin

    Parameters:
    -----------
    companies : list of dict
        Each dict has: name, revenue, ebitda, market_cap, net_debt

    Returns:
    --------
```

```
    None (prints formatted table)
    """
    # Your code here
    pass

# Test data
comps = [
    {"name": "TechCo A", "revenue": 1000, "ebitda": 200, "market_cap":
1500, "net_debt": 100},
    {"name": "TechCo B", "revenue": 1200, "ebitda": 250, "market_cap":
2000, "net_debt": 200},
    {"name": "TechCo C", "revenue": 800, "ebitda": 150, "market_cap":
1200, "net_debt": 50},
]

create_comps_table(comps)
```

Expected output:

```
Comparable Companies Analysis:

Company       Revenue  EBITDA  EV        EV/Rev  EV/EBITDA  EBITDA Margin
------------------------------------------------------------------------
TechCo A      $1,000   $200    $1,600    1.6x    8.0x       20.0%
TechCo B      $1,200   $250    $2,200    1.8x    8.8x       20.8%
TechCo C      $800     $150    $1,250    1.6x    8.3x       18.8%
------------------------------------------------------------------------
Median        $1,000   $200    $1,600    1.6x    8.3x       20.0%
```

Hint:

- EV = Market Cap + Net Debt
- Use f-strings for formatting
- Calculate median using sorted()[len//2]

Solutions

Complete solutions are available in `Module_02_Python_Fundamentals/solutions.py`.

Try solving these yourself first! The learning happens when you struggle through the problem. If you get stuck:

1. **Re-read the hints** - they contain the formula/approach
2. **Break it down** - solve one small part at a time
3. **Print intermediate values** - see what's happening
4. **Check the solutions** - but try first!

Key Takeaways

By completing this module, you now understand:

✅ **Variables**: Store and manipulate financial data
✅ **Numbers**: Perform calculations like Excel formulas
✅ **Strings**: Work with company names, tickers, labels
✅ **Lists**: Handle time series (revenues, cash flows)
✅ **Dictionaries**: Structure financial data (company financials)
✅ **Functions**: Create reusable financial formulas (WACC, NPV, IRR)
✅ **Loops**: Automate projections and repetitive tasks
✅ **Conditionals**: Make investment decisions based on criteria
✅ **Classes**: Build sophisticated, reusable financial models

You're no longer a Python beginner! You can now build financial models that would take hours in Excel.

What's Next?

Continue to: `Module_03_Data_Analysis/01_Pandas_for_Finance.md`

In Module 3, you'll learn to:

- Load financial data from Excel, CSV, and APIs
- Clean and transform messy data
- Perform advanced analysis with Pandas
- Create professional financial tables and reports

Before moving on, make sure you:

- ✅ Completed all examples in this module
- ✅ Attempted the practice exercises
- ✅ Understand functions, loops, and conditionals
- ✅ Can create a simple Company class

Estimated Time: 3-4 hours
Prerequisites: Module 01 (Setup) completed
Difficulty: ⭐⭐ Beginner-Intermediate

Congratulations, Mauricio! You've mastered Python fundamentals for finance! 🎉

This is the foundation for everything you'll build at PE Club. Keep going!