PYTHON CRASH COURSE

for Operations & Supply Chain Management

BCOR 440 & IE 425 • Gannon University

Spring 2026

A Beginner-Friendly Guide to Python for Business Students

No prior coding experience required!

Welcome: Why Python?

You might be thinking: “I’m a business student, not a computer science major. Why do I need to learn Python?” Great question. Here’s the honest answer:

- **Python is the #1 language used in business analytics.** Companies like Amazon, Walmart, and Starbucks use Python to forecast demand, optimize inventory, and streamline supply chains.
- **It’s the easiest programming language to learn.** Python reads almost like English. If you can write a formula in Excel, you can learn Python.
- **It makes you stand out.** Listing “Python” on your resume immediately signals analytical capability to employers—regardless of your major.
- **AI runs on Python.** ChatGPT, supply chain optimization tools, predictive analytics—they’re all built on Python. Understanding the basics helps you work smarter with these tools.

🎯 What You’ll Be Able to Do After This Guide

• Write simple Python scripts to solve operations problems
• Use Python to analyze data in ways Excel can’t easily handle
• Calculate key OSCM metrics (EOQ, forecasts, lead times) with code
• Understand how GenAI + Python can supercharge your analysis
• Add “Python” to your LinkedIn skills with confidence

Section 1: Getting Started

1.1 — Setting Up (No Installation Needed!)

The fastest way to start writing Python is Google Colab—a free, browser-based tool from Google. No downloads, no installation headaches.

1. Go to colab.research.google.com
2. Sign in with your Google account
3. Click "+ New Notebook"
4. Start typing code in the cell and press Shift + Enter to run it

> **Why Google Colab?**
> • It's free and runs in your browser
> • All the major libraries (pandas, numpy, matplotlib) are pre-installed
> • You can share notebooks like Google Docs
> • Your work saves automatically to Google Drive

1.2 — Your First Python Code

Let's start with the classic. Type this into your Colab cell and hit Shift + Enter:

```
print("Hello, Supply Chain World!")
```

You should see the text printed below the cell. Congratulations—you just wrote your first Python program!

Now let's do something more useful:

```
# Calculate total inventory cost
holding_cost = 2.50      # $ per unit per year
units_in_stock = 1000
total_cost = holding_cost * units_in_stock

print(f"Annual holding cost: ${total_cost:,.2f}")
```

Output: `Annual holding cost: $2,500.00`

> **Key Things to Notice**
> • Lines starting with # are comments—Python ignores them (but humans need them!)
> • Variable names use underscores: holding_cost, not holdingCost

- f"..." is an f-string—it lets you embed variables inside text
- :,.2f formats numbers with commas and 2 decimal places

Section 2: Python Essentials (The Stuff You Actually Need)

You don't need to learn all of Python. Here are the building blocks that matter for OSCM work.

2.1 — Variables & Data Types

Variables are containers for information. Think of them like labeled boxes in a warehouse.

```
# Numbers (integers and decimals)
order_quantity = 500          # integer
unit_cost = 12.75             # float (decimal)

# Text (strings)
supplier_name = "Acme Corp"
warehouse = 'Distribution Center 3'

# True/False (booleans)
is_backordered = True
meets_quality = False

# Lists (ordered collections)
monthly_demand = [120, 135, 98, 145, 160, 142]

# Dictionaries (labeled data)
product = {
    "name": "Widget A",
    "sku": "WA-1001",
    "price": 24.99,
    "in_stock": True
}
```

2.2 — Basic Math Operations

Python handles math just like a calculator, but way more powerful:

Operation	Symbol	Example	Result
Addition	+	50 + 30	80
Subtraction	-	100 - 25	75
Multiplication	*	12 * 5	60
Division	/	100 / 3	33.33
Integer Division	//	100 // 3	33

Remainder (Modulo)	%	100 % 3	1
Exponent (Power)	**	2 ** 10	1024

2.3 — Lists: Your Go-To Data Structure

Lists are everywhere in OSCM. Demand data, supplier ratings, lead times—they're all lists.

```
demand = [120, 135, 98, 145, 160, 142]

# Access items (Python counts from 0!)
print(demand[0])       # First item: 120
print(demand[-1])      # Last item: 142

# Useful list tricks
print(len(demand))     # How many items: 6
print(sum(demand))     # Total: 800
print(max(demand))     # Highest: 160
print(min(demand))     # Lowest: 98

# Average demand
avg = sum(demand) / len(demand)
print(f"Average demand: {avg:.1f}")  # 133.3
```

> ⚠ **Common Gotcha: Indexing Starts at 0**
> The first item in a list is demand[0], not demand[1].
> This trips up almost everyone at first. You'll get used to it!

2.4 — If/Else: Making Decisions

Business decisions are all about conditions: If demand is high, order more. If quality drops, stop the line.

```
inventory_level = 45
reorder_point = 50

if inventory_level < reorder_point:
    print("ALERT: Time to reorder!")
    print(f"Current stock: {inventory_level} units")
elif inventory_level < reorder_point * 1.2:
    print("Getting low - monitor closely")
else:
    print("Stock levels are healthy")
```

> **Indentation Matters!**
> Python uses indentation (4 spaces) instead of curly braces.
> The indented lines under “if” only run when the condition is True.
> This actually makes code easier to read once you get used to it.

2.5 — Loops: Doing Things Repeatedly

Loops save you from doing the same thing over and over. Perfect for processing batches of data.

```
# Check each supplier’s delivery time
lead_times = {"Supplier A": 5, "Supplier B": 12, "Supplier C": 3, "Supplier D": 8}

for supplier, days in lead_times.items():
    if days > 7:
        status = "SLOW - consider alternatives"
    else:
        status = "On track"
    print(f"{supplier}: {days} days - {status}")
```

Output:

```
Supplier A: 5 days - On track
Supplier B: 12 days - SLOW - consider alternatives
Supplier C: 3 days - On track
Supplier D: 8 days - SLOW - consider alternatives
```

2.6 — Functions: Reusable Code Blocks

Functions let you package code so you can reuse it. Write once, use many times.

```
def calculate_eoq(annual_demand, order_cost, holding_cost):
    """Calculate Economic Order Quantity"""
    eoq = (2 * annual_demand * order_cost / holding_cost) ** 0.5
    return round(eoq)

# Use it for different products
eoq_widget = calculate_eoq(10000, 50, 2)
eoq_gadget = calculate_eoq(5000, 75, 3)

print(f"Widget EOQ: {eoq_widget} units")
print(f"Gadget EOQ: {eoq_gadget} units")
```

Output: `Widget EOQ: 707 units | Gadget EOQ: 500 units`

Section 3: Pandas — Excel on Steroids

If Python is the engine, pandas is the steering wheel for data analysis. It lets you work with spreadsheet-style data inside Python. Think of it as Excel—but faster, more powerful, and able to handle millions of rows without crashing.

3.1 — Creating and Viewing Data

```
import pandas as pd

# Create a DataFrame (like a spreadsheet)
data = {
    "Product": ["Widget A", "Widget B", "Gadget X", "Gadget Y"],
    "Monthly_Demand": [500, 300, 800, 150],
    "Unit_Cost": [12.50, 18.00, 8.75, 45.00],
    "Lead_Time_Days": [5, 12, 3, 8]
}

df = pd.DataFrame(data)
print(df)
```

Output:

```
    Product  Monthly_Demand  Unit_Cost  Lead_Time_Days
0  Widget A             500      12.50               5
1  Widget B             300      18.00              12
2  Gadget X             800       8.75               3
3  Gadget Y             150      45.00               8
```

3.2 — Quick Analysis

```
# Summary statistics (like Excel's Data Analysis)
print(df.describe())

# Filter: Which products have lead time > 7 days?
slow_items = df[df["Lead_Time_Days"] > 7]
print("\nSlow-delivery products:")
print(slow_items)

# Add a new column
df["Monthly_Cost"] = df["Monthly_Demand"] * df["Unit_Cost"]
print(f"\nTotal monthly spend: ${df['Monthly_Cost'].sum():,.2f}")
```

3.3 — Reading Real Data Files

In the real world, you'll work with CSV and Excel files—not manually typed data:

```
# Read from CSV
df = pd.read_csv("sales_data.csv")

# Read from Excel
df = pd.read_excel("inventory_report.xlsx")

# Quick peek at the data
print(df.head())          # First 5 rows
print(df.shape)           # (rows, columns)
print(df.columns.tolist())  # Column names
```

When to Use Pandas vs. Excel

- Small dataset, one-time analysis → Excel is fine
- Large dataset (10,000+ rows) → Pandas is faster
- Repetitive analysis → Pandas (write once, run anytime)
- Combining multiple files → Pandas (a few lines of code vs. hours of copy-paste)

Section 4: Hands-On OSCM Examples

Here's where it gets fun. Let's solve real operations problems with Python.

Example 1: EOQ Calculator

The Economic Order Quantity formula balances ordering costs against holding costs. Here it is in Python:

```
import math

def eoq_analysis(product_name, annual_demand, order_cost, holding_cost_pct, unit_cost):
    """Complete EOQ analysis for a product"""
    holding_cost = holding_cost_pct * unit_cost
    eoq = math.sqrt(2 * annual_demand * order_cost / holding_cost)
    orders_per_year = annual_demand / eoq
    total_order_cost = orders_per_year * order_cost
    total_holding_cost = (eoq / 2) * holding_cost
    total_cost = total_order_cost + total_holding_cost

    print(f"\n=== EOQ Analysis: {product_name} ===")
    print(f"  EOQ: {eoq:,.0f} units")
    print(f"  Orders/year: {orders_per_year:,.1f}")
    print(f"  Total ordering cost: ${total_order_cost:,.2f}")
    print(f"  Total holding cost:  ${total_holding_cost:,.2f}")
    print(f"  Total annual cost:   ${total_cost:,.2f}")

# Run it!
eoq_analysis("Laptop Batteries", 12000, 75, 0.25, 45)
eoq_analysis("USB Cables", 50000, 25, 0.20, 3.50)
```

Example 2: Simple Moving Average Forecast

Forecasting is about using past data to predict the future. A moving average smooths out random fluctuations.

```
import pandas as pd

# Monthly sales data
sales = [220, 250, 180, 270, 230, 260, 290, 240, 310, 275, 320, 300]
months = ["Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec"]

df = pd.DataFrame({"Month": months, "Sales": sales})
```

```
# 3-month moving average
df["3M_Avg"] = df["Sales"].rolling(window=3).mean().round(1)

# Forecast next month using last 3 months
forecast = df["Sales"].tail(3).mean()
print(f"\nJanuary forecast: {forecast:.0f} units")
print(df.to_string(index=False))
```

Example 3: Supplier Scorecard

Evaluating suppliers on multiple criteria is a classic OSCM task. Python makes it easy to weigh and rank them:

```
import pandas as pd

suppliers = pd.DataFrame({
    "Supplier": ["Alpha Inc", "Beta Corp", "Gamma LLC", "Delta Co"],
    "Quality": [92, 88, 95, 85],       # out of 100
    "Delivery": [90, 95, 80, 92],       # on-time %
    "Price": [85, 78, 90, 95],          # competitiveness
    "Flexibility": [80, 85, 75, 88]
})

# Weighted scoring
weights = {"Quality": 0.35, "Delivery": 0.30, "Price": 0.20, "Flexibility": 0.15}

suppliers["Weighted_Score"] = (
    suppliers["Quality"] * weights["Quality"] +
    suppliers["Delivery"] * weights["Delivery"] +
    suppliers["Price"] * weights["Price"] +
    suppliers["Flexibility"] * weights["Flexibility"]
).round(1)

# Rank them
suppliers = suppliers.sort_values("Weighted_Score", ascending=False)
suppliers["Rank"] = range(1, len(suppliers) + 1)
print(suppliers.to_string(index=False))
```

Example 4: ABC Inventory Classification

ABC analysis classifies inventory items by their dollar impact. A-items (top 80% of value) get the most attention.

```
import pandas as pd

inventory = pd.DataFrame({
    "SKU": ["A101","B202","C303","D404","E505","F606","G707","H808"],
    "Annual_Demand": [5000, 300, 12000, 100, 8000, 50, 1500, 2000],
    "Unit_Cost": [50, 200, 5, 500, 15, 800, 25, 30]
})

# Calculate annual dollar usage
inventory["Annual_Value"] = inventory["Annual_Demand"] * inventory["Unit_Cost"]
inventory = inventory.sort_values("Annual_Value", ascending=False)

# Cumulative percentage
total = inventory["Annual_Value"].sum()
inventory["Cum_Pct"] = (inventory["Annual_Value"].cumsum() / total * 100).round(1)

# Classify
inventory["Class"] = inventory["Cum_Pct"].apply(
    lambda x: "A" if x <= 80 else ("B" if x <= 95 else "C")
)

print(inventory.to_string(index=False))
print(f"\nA items: {(inventory['Class']=='A').sum()}")
print(f"B items: {(inventory['Class']=='B').sum()}")
print(f"C items: {(inventory['Class']=='C').sum()}")
```

Section 5: Data Visualization

A picture is worth a thousand spreadsheet rows. Python's matplotlib library creates professional charts with just a few lines of code.

5.1 — Demand Trend Chart

```
import matplotlib.pyplot as plt

months = ["Jan","Feb","Mar","Apr","May","Jun",
          "Jul","Aug","Sep","Oct","Nov","Dec"]
demand = [120, 135, 98, 145, 160, 142, 170, 155, 180, 165, 190, 200]
forecast = [None, None, None] + [
    round(sum(demand[i-3:i])/3) for i in range(3, len(demand))
]

plt.figure(figsize=(10, 5))
plt.plot(months, demand, "o-", color="#2E8B8B", label="Actual Demand", linewidth=2)
plt.plot(months, forecast, "s--", color="#E8913A", label="3-Mo Moving Avg", linewidth=2)
plt.title("Monthly Demand vs. Forecast", fontsize=14, fontweight="bold")
plt.xlabel("Month")
plt.ylabel("Units")
plt.legend()
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
```

5.2 — Pareto Chart (ABC Analysis Visual)

```
import matplotlib.pyplot as plt

# Using data from the ABC example above
fig, ax1 = plt.subplots(figsize=(10, 5))

# Bar chart for annual value
ax1.bar(inventory["SKU"], inventory["Annual_Value"],
        color="#2E8B8B", alpha=0.7)
ax1.set_ylabel("Annual Dollar Value ($)", color="#2E8B8B")

# Line chart for cumulative %
ax2 = ax1.twinx()
ax2.plot(inventory["SKU"], inventory["Cum_Pct"],
         "o-", color="#E8913A", linewidth=2)
ax2.set_ylabel("Cumulative %", color="#E8913A")
```

```
ax2.axhline(y=80, color="red", linestyle="--", alpha=0.5, label="80% line")

plt.title("ABC Analysis - Pareto Chart", fontsize=14, fontweight="bold")
plt.tight_layout()
plt.show()
```

Chart Tips for Presentations

- Always include a clear title that tells the story
- Label your axes (include units!)
- Use plt.savefig('chart.png', dpi=300) to save high-res images for slides
- Keep colors consistent across related charts

Section 6: GenAI + Python in OSCM

Generative AI tools like ChatGPT and Claude are game-changers for working with Python—especially if you're new to coding. But there's a catch: AI is a powerful assistant, not a replacement for your brain. Here's how to use them effectively.

6.1 — How GenAI Helps You Code

Use Case	What to Ask GenAI	OSCM Example
Generate starter code	"Write Python code to calculate EOQ"	Quickly get a working EOQ function you can modify
Debug errors	"Why am I getting this error?" (paste the error)	Fix a pandas merge that produced unexpected results
Explain code	"What does this line do?"	Understand a forecasting function from a colleague
Optimize analysis	"Make this code faster / cleaner"	Speed up a slow inventory classification script
Learn new techniques	"How do I create a Pareto chart in Python?"	Pick up visualization skills for presentations
Data cleaning	"Help me clean this messy CSV data"	Fix inconsistent date formats in shipment records

6.2 — The Smart Way to Use GenAI for OSCM Analysis

Here's a workflow that maximizes AI's strengths while keeping you in the driver's seat:

Step 1: Start with a Clear Prompt

Be specific about what you need. Vague prompts get vague code.

```
# BAD PROMPT:
"Write code for inventory management"

# GOOD PROMPT:
"Write a Python function that calculates the reorder point
 given average daily demand, lead time in days, and desired
 service level (z-score). Include safety stock calculation.
 Use clear variable names and add comments."
```

Step 2: Review and Understand the Output

Never copy-paste AI code blindly. Always ask yourself:

- Does this formula match what we learned in class?
- Do the variable names make sense?
- Can I explain what each section does?

Step 3: Test with Known Answers

Run the code with data where you already know the answer. If EOQ should be 707 units and the code returns 707, you're good.

Step 4: Modify for Your Specific Needs

AI gives you a starting point. The real value comes from customizing it for your specific analysis, data, and business context.

> ⚠ **GenAI Pitfalls to Watch For**
> • Hallucinated functions: AI may invent library functions that don't exist. Always test.
> • Outdated syntax: AI sometimes uses old Python/library versions. Check documentation.
> • Wrong assumptions: AI doesn't know your data. Verify column names, data types, and units.
> • Over-complexity: AI may write 50 lines when 10 will do. Simpler is usually better.
> • Missing context: AI doesn't understand YOUR business. You provide the strategic judgment.

6.3 — Real-World GenAI + Python Scenarios

Scenario A: Demand Forecasting

Ask GenAI: "Write Python code using pandas to apply exponential smoothing to monthly sales data. Use an alpha of 0.3. Include a chart comparing actual vs. forecast."

Then: Check that the smoothing formula is correct, swap in your real data, and interpret the results in your business context.

Scenario B: Inventory Optimization

Ask GenAI: "Create a Python script that reads an inventory CSV file, performs ABC analysis, and outputs a summary showing how many items are in each class and their percentage of total value."

Then: Verify the classification thresholds (80/95) match what we use in class, validate with a manual check on a few items.

Scenario C: Process Analysis

Ask GenAI: "Write a Python simulation of a 4-step production process. Each step has different processing times (normally distributed). Calculate throughput, bottleneck identification, and average wait times."

Then: Compare the bottleneck output to your Theory of Constraints analysis. Does the simulation match your manual calculation?

Scenario D: Data Cleaning

Ask GenAI: "I have a CSV with messy supplier data—inconsistent date formats, missing values, and duplicate entries. Write Python code using pandas to clean this data: standardize dates to YYYY-MM-DD, fill missing lead times with the median, and remove duplicates."

Then: Spot-check the cleaned data. Did it correctly identify duplicates? Are the filled values reasonable?

Section 7: Python Quick Reference Cheat Sheet

Tear this page out (or bookmark it). These are the commands you'll use most often.

Essential Python

What You Want to Do	Python Code
Print something	print("Hello!")
Create a variable	x = 42
Create a list	data = [10, 20, 30]
Get list length	len(data)
Sum a list	sum(data)
Average of a list	sum(data) / len(data)
Loop through items	for item in data:
Check a condition	if x > 10:
Define a function	def my_func(x):
Import a library	import pandas as pd
Square root	x ** 0.5 or math.sqrt(x)
Round a number	round(3.14159, 2) → 3.14

Essential Pandas

What You Want to Do	Pandas Code
Read a CSV file	pd.read_csv("file.csv")
Read an Excel file	pd.read_excel("file.xlsx")
View first 5 rows	df.head()
Get summary stats	df.describe()
Filter rows	df[df["column"] > 100]
Sort by column	df.sort_values("column")
Add new column	df["new"] = df["a"] * df["b"]
Group and summarize	df.groupby("category").mean()
Count values	df["column"].value_counts()
Save to CSV	df.to_csv("output.csv", index=False)
Moving average	df["col"].rolling(window=3).mean()

Remove duplicates	df.drop_duplicates()

Essential Matplotlib

Chart Type	Code
Line chart	plt.plot(x, y, "o-")
Bar chart	plt.bar(x, y)
Histogram	plt.hist(data, bins=10)
Scatter plot	plt.scatter(x, y)
Add title	plt.title("My Chart")
Add labels	plt.xlabel("X") / plt.ylabel("Y")
Add legend	plt.legend()
Show grid	plt.grid(True)
Save figure	plt.savefig("chart.png", dpi=300)
Display chart	plt.show()

Section 8: Where to Go from Here

You now have a solid foundation. Here's how to keep building:

Immediate Next Steps

1. Practice the examples in this guide using Google Colab—don't just read, run the code!
2. Modify the examples with different numbers and see what changes
3. Try loading a real CSV dataset and exploring it with pandas
4. Use GenAI to help when you get stuck—then make sure you understand the solution

Free Resources to Keep Learning

Resource	Best For	URL
Google Colab Tutorials	Hands-on practice (free)	colab.research.google.com
Kaggle Learn (Python)	Interactive lessons + datasets	kaggle.com/learn/python
Python for Data Analysis (book)	Deep dive into pandas	By Wes McKinney (library/online)
Real Python	Clear tutorials on specific topics	realpython.com
Kaggle Datasets	Real OSCM datasets to practice with	kaggle.com/datasets

For Your Resume & LinkedIn

Once you've completed the exercises in this guide, you can confidently add these to your LinkedIn skills:

- Python (Basic)
- Data Analysis with pandas
- Data Visualization (matplotlib)
- Business Analytics

> 🚀 **The #1 Rule of Learning to Code**
> The only way to learn Python is by DOING Python.
> Reading this guide is step one. Running the code is step two.
> Breaking the code (on purpose!) is step three.
> Fixing what you broke? That's when the real learning happens.

— End of Python Crash Course —

Good luck, and remember: every expert was once a beginner.