Lesson 02: Data Structures

Data Science with Python – BSc Course

Data Science Program

45 Minutes

Learning Objectives

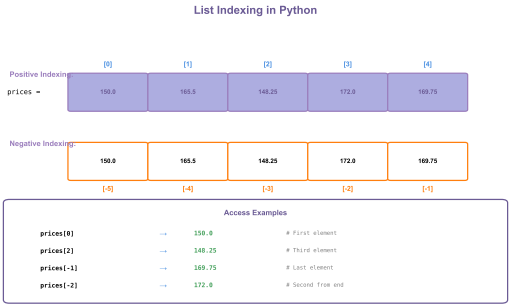
After this lesson, you will be able to:

- Create and manipulate Python lists
- Access elements using indexing and slicing
- Build dictionaries for key-value data storage
- Apply list comprehensions for efficient data processing

Finance Application: Store portfolio holdings as lists and dictionaries.

Data structures are containers for organizing information

List Indexing



Creating Lists:
`prices = [185, 190, 188, 195]`

Accessing Elements:
`prices[0]` → 185 (first)
`prices[-1]` → 195 (last)
`prices[1]` → 190 (second)

Remember:
Python indexing starts at 0!

Negative indices count from the end: -1 is last element

Slicing Notation

Slice Syntax: `list[start:end:step]`

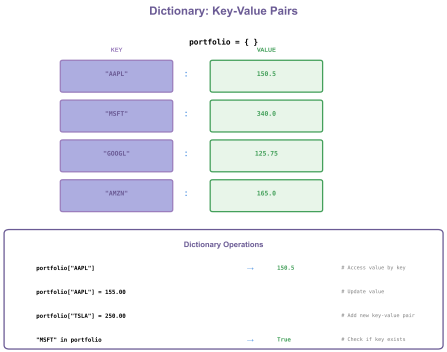
```
prices = [185, 190, 188, 195, 182]
```

`prices[1:4]` → `[190, 188, 195]`
`prices[:3]` → `[185, 190, 188]`
`prices[2:]` → `[188, 195, 182]`
`prices[::2]` → `[185, 188, 182]`

Key: End index is exclusive

Slicing creates a new list – original unchanged

Dictionary Structure



Key-Value Pairs:

```
portfolio = {
    "AAPL":  50,
    "MSFT":  30,
    "GOOGL":  20
}
```

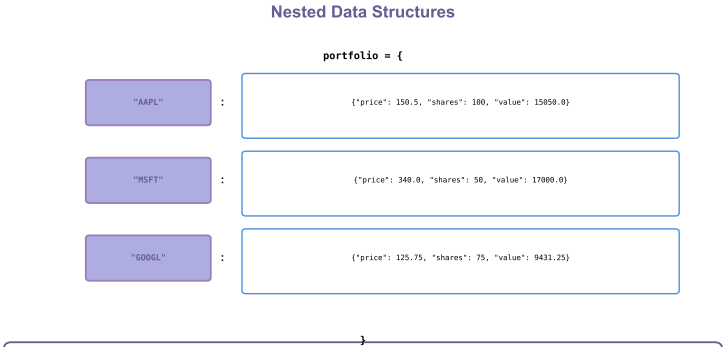
Access by Key:

`portfolio["AAPL"]` → 50

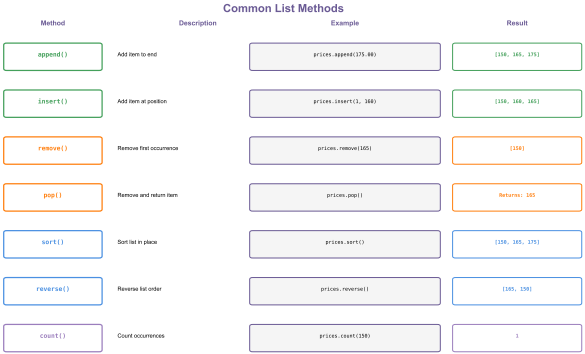
```
portfolio.keys()
portfolio.values()
```

Dictionaries provide O(1) lookup – very fast access

Nested Data Structures



List Methods



Adding Elements:

```
prices.append(200)
prices.insert(0, 180)
```

Removing:

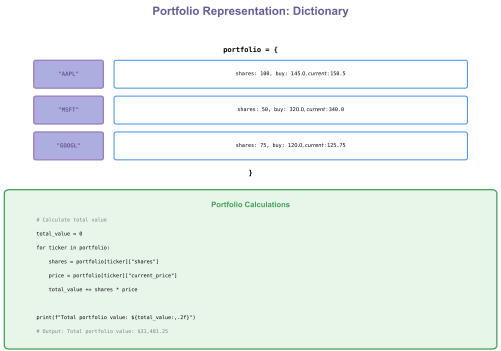
```
prices.remove(188)
prices.pop() – removes last
```

Sorting:

```
prices.sort()
prices.reverse()
```

Methods modify the list in-place (except sorted())

Portfolio as Dictionary



Portfolio Dictionary:

```
prices = {
    "AAPL":  185.50,
    "MSFT":  378.20,
    "GOOGL":  141.80
}
```

Calculate Total:

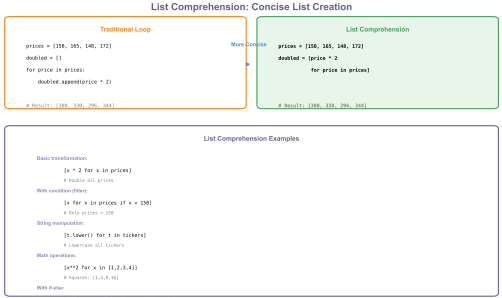
```
total = sum(prices.values())
```

Check Existence:

`"AAPL" in prices` → `True`

Dictionaries are ideal for ticker-to-data mappings

List Comprehension



Traditional Loop:

```
returns = []
for p in prices:
    returns.append(p * 1.05)
```

List Comprehension:

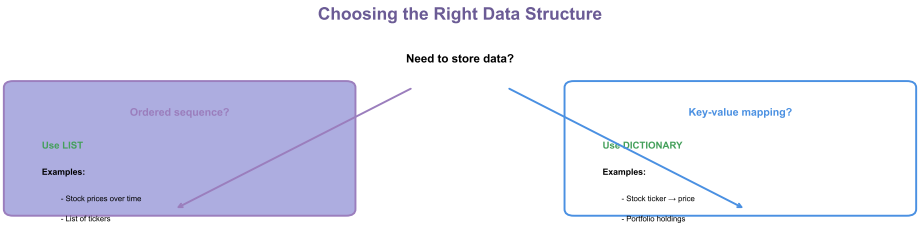
```
returns = [p * 1.05 for p in prices]
```

With Condition:

```
high = [p for p in prices if p > 190]
```

Comprehensions are more Pythonic and often faster

Choosing the Right Structure



Feature Comparison

List	Feature	Dictionary
`Ordered`	Order	`Unordered`
`prices[0]`	Access	`portfolio["AAPL"]`
`O(n) search`	Speed	`O(1) lookup`
`Duplicates OK`	Duplicates	`Unique keys`
`Integer indices`	Keys	`Any immutable`

Hands-on Exercise (25 min)

Build a portfolio tracker:

1. Create a list of stock tickers:
   `tickers = ["AAPL", "MSFT", "GOOGL", "AMZN"]`
2. Create a dictionary with shares owned:
   `shares = {"AAPL": 50, "MSFT": 30, ...}`
3. Create a dictionary with current prices
4. Calculate portfolio value using list comprehension:
   `values = [shares[t] * prices[t] for t in tickers]`
5. Find total portfolio value: `sum(values)`
6. Filter stocks worth more than $5000

Save your work – we'll add more features next lesson

Lesson Summary

Key Takeaways:

- Lists store ordered sequences (accessed by index)
- Dictionaries store key-value pairs (accessed by key)
- Slicing extracts portions: `list[start:end]`
- List comprehensions create lists efficiently
- Choose structure based on access pattern

Next Lesson: Control Flow (if/else, loops)

Data structures + control flow = programming logic