Python for Cheminformatics & Bioinformatics

Combined Lessons & Labs

Nirajan Bhattarai

AI-Driven Drug Development Training

February 2026

Course Overview

1. Data Types in Drug Discovery
2. Python Basics (Lessons 1–6B)
   - Lesson 1: Variables & Data Types
   - Lesson 2: Operators
   - Lesson 3: Strings
   - Lesson 4: Conditionals
   - Lesson 5: Loops
   - Lesson 6: Functions
   - Lesson 6B: Error Handling
3. Collections & Advanced Python (Lessons 7–12)
   - Lesson 7: Lists
   - Lesson 7B: Tuples & Sets
   - Lesson 8: List Comprehensions
   - Lesson 9: Dictionaries
   - Lesson 10: File Handling
   - Lesson 11: NumPy
   - Lesson 11B: Pandas
   - Lesson 12: JSON & R

Learning Objectives

By the end of this course, you will be able to:

1. Write Python code for basic data manipulation
2. Work with molecular data (SMILES, formulas, properties)
3. Process biological sequences (DNA, RNA, protein)
4. Use control flow (conditionals, loops) for data filtering
5. Organize data using lists, dictionaries, and sets
6. Create reusable functions for cheminformatics tasks
7. Read/write files (CSV, FASTA, JSON)
8. Use NumPy arrays and Pandas DataFrames
9. Apply concepts to Rosalind bioinformatics problems

Prerequisites & Setup

Prerequisites:

- Basic computer literacy
- No prior programming experience required
- Interest in drug discovery / life sciences

Software Setup:

- Python 3.8+ installed
- IDE: VS Code, PyCharm, or Jupyter Notebook
- Libraries: `pip install numpy pandas rdkit`

Resources:

- Rosalind.info – Bioinformatics problems
- ChEMBL – Bioactivity database
- PubChem – Chemical information

Data Types You'll Encounter

Before we start coding, let's understand the data types used in drug discovery:

Cheminformatics:

- SMILES – Molecular structures
- Molecular Descriptors (MW, LogP)
- Activity Data (IC50, pIC50)
- Lipinski Properties

Bioinformatics:

- DNA Sequences (A, T, G, C)
- RNA Sequences (A, U, G, C)
- Protein Sequences (amino acids)
- FASTA Format

Understanding these data types is essential for the exercises in this course.

SMILES: Simplified Molecular Input Line Entry System

What is SMILES?
A text-based notation for representing chemical structures as strings.

Key Rules:

- Atoms: C, N, O, S, P, F, Cl, Br, I (organic subset)
- Bonds: single (default), double (=), triple (#), aromatic (:)
- Rings: Numbers indicate ring closures (e.g., `c1ccccc1` = benzene)
- Branches: Parentheses for side chains (e.g., `CC(C)C` = isobutane)

Examples:

`CCO`	Ethanol
`CC(=O)O`	Acetic acid
`c1ccccc1`	Benzene
`CC(=O)Oc1ccccc1C(=O)O`	Aspirin

Molecular Descriptors & Activity Data

Key Molecular Properties:

Property	Description	Typical Range
MW	Molecular Weight (Da)	150–500 Da
LogP	Lipophilicity	-2 to 5
HBD	H-bond Donors	0–5
HBA	H-bond Acceptors	0–10

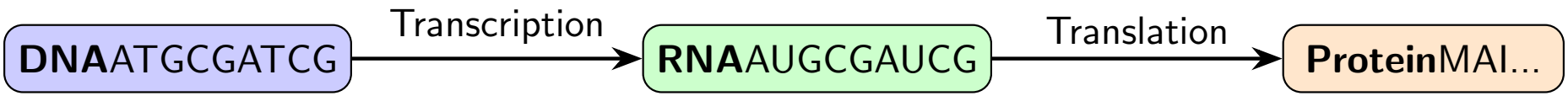
pIC50 Conversion:

$$\text{pIC50} = -\log_{10}(\text{IC50}_{\text{M}}) = 9 - \log_{10}(\text{IC50}_{\text{nM}})$$

Activity Classification:	$\text{pIC50} \geq 8$	$\text{IC50} \leq 10$ nM	Highly Active
	6–8	10–1000 nM	Active
	< 6	> 1000 nM	Inactive

DNA, RNA & Protein Sequences

DNA: A, T, G, C (double helix, base pairing: A-T, G-C)
RNA: A, U, G, C (single strand, T $\rightarrow$ U)
Protein: 20 amino acids encoded by codons



Key Operations:

- Transcribe: DNA $\rightarrow$ RNA (replace T with U)
- Reverse Complement: A$\leftrightarrow$T, G$\leftrightarrow$C, then reverse
- GC Content: (G + C) / total $\times$ 100%

Lesson 1: Learning Objectives

Learning Objectives:

- Understand how variables store data in memory
- Identify Python's core data types (int, float, str, bool)
- Perform type conversions between data types

Applications:

- Store compound properties (MW, LogP, SMILES)
- Represent bioactivity measurements (IC50, pIC50)
- Handle DNA/RNA sequence data

Lesson 1: Variables & Data Types

Concept: Variables store data in memory.

Python data types: `int`, `float`, `str`, `bool`, `NoneType`

Type Conversion: `int()`, `float()`, `str()`, `bool()`

Drug Discovery Scenarios:

- Compound Info (name, MW, LogP, SMILES)
- Bioactivity Data (IC50, Ki, pIC50)
- DNA/RNA Sequences (nucleotide strings)

Lesson 1 Code Example

```python
# Compound Info
compound_name = "Aspirin"
mw = 180.16   # Molecular Weight (Da)
logP = 1.19   # Lipophilicity
smiles = "CC(=O)OC1=CC=CC=C1C(=O)O"

# Bioactivity Data
ic50_nM = 5.2   # IC50 in nanomolar
pic50 = 8.28    # -log10(IC50 in M)
is_active = True

# DNA Sequence
dna_seq = "ATGCGATCGATCG"
seq_length = len(dna_seq)

# Type Conversion
ic50_str = str(ic50_nM)
mw_int = int(mw)

print(f"{compound_name}: MW={mw}, pIC50={pic50}")
```

Lab 1: Variables & Data Types

Objective: Practice storing and converting molecular and sequence data.

Exercise 1.1 – Compound Data Storage

Create variables for a drug compound:

- Name (string): "Ibuprofen"
- SMILES (string): "CC(C)CC1=CC=C(C=C1)C(C)C(=O)O"
- Molecular weight (float): 206.28
- pIC50 (float): 6.1
- Is active (bool): True if pIC50 $\geq$ 6.0

Print all values using f-strings.

Lab 1: Variables & Data Types (cont.)

Exercise 1.2 – DNA Sequence

Store a DNA sequence: "ATGCGATCGATCGATCGATCG"
Calculate and print:

- Sequence length
- Number of adenines (A)
- Number of thymines (T)

Exercise 1.3 – Type Conversion

Given IC50 = "5.2" (string), convert to float and calculate pIC50.
Store the result as both float and formatted string (2 decimals).

Lesson 2: Learning Objectives

Learning Objectives:

- Use arithmetic operators for scientific calculations
- Apply comparison operators to filter data
- Combine conditions using logical operators

Applications:

- Convert IC50 to pIC50 ($-\log_{10}$)
- Check Lipinski Rule of Five compliance
- Calculate GC content in DNA sequences

Lesson 2: Operators

Arithmetic: `+`, `-`, `*`, `/`, `%`, `**`

Comparison: `>`, `<`, `==`, `!=`, `>=`, `<=`

Logical: `and`, `or`, `not`

Drug Discovery scenarios: Calculate pIC50, check Lipinski rules, filter active compounds

Lesson 2 Code Example

```python
import math

# IC50 to pIC50 conversion
ic50_nM = 10.0  # nanomolar
ic50_M = ic50_nM * 1e-9  # convert to molar
pic50 = -math.log10(ic50_M)  # pIC50 = 8.0
print(f"IC50: {ic50_nM} nM -> pIC50: {pic50:.2f}")

# Lipinski Rule of Five checks
mw, logP, hbd, hba = 450, 3.5, 2, 6
lipinski_ok = (mw <= 500) and (logP <= 5) and (hbd <= 5) and
    (hba <= 10)
print(f"Passes Lipinski: {lipinski_ok}")

# GC Content calculation
seq = "ATGCGCGCTA"
gc_count = seq.count("G") + seq.count("C")
gc_percent = (gc_count / len(seq)) * 100
print(f"GC Content: {gc_percent:.1f}%")
```

Lab 2: Operators

Objective: Apply operators for molecular calculations and filtering.

Exercise 2.1 – IC50 Conversion

Convert IC50 values from nM to pIC50:
IC50 values: 10.0, 100.0, 1000.0 (nM)
Formula: $\text{pIC50} = 9 - \log_{10}(\text{IC50\_nM})$

Exercise 2.2 – Lipinski Check

Given: MW=450, LogP=3.5, HBD=2, HBA=8
Check if compound passes Rule of Five:
$(\text{MW} \leq 500)$ AND $(\text{LogP} \leq 5)$ AND $(\text{HBD} \leq 5)$ AND $(\text{HBA} \leq 10)$

Lab 2: Operators (cont.)

Exercise 2.3 – GC Content (Rosalind)

Calculate GC content percentage for: "AGCTATAG"
Formula: GC% = (G + C) / total × 100

Exercise 2.4 – Activity Classification

Given pIC50 = 7.2, determine if compound is:

- "Highly potent" (pIC50 $\geq$ 8)
- "Potent" (pIC50 $\geq$ 7)
- "Moderate" (pIC50 $\geq$ 6)
- "Weak" (pIC50 $<$ 6)

Lesson 3: Learning Objectives

Learning Objectives:

- Manipulate strings using indexing and slicing
- Apply string methods for text processing
- Parse and transform sequence data

Applications:

- Transcribe DNA to RNA (T $\rightarrow$ U)
- Generate reverse complement sequences
- Parse SMILES for molecular features

Lesson 3: Strings

Strings store text – essential for sequences and SMILES.

Methods: indexing/slicing, `len()`, `upper()`, `lower()`, `replace()`, `split()`, `count()`, `find()`

Drug Discovery scenarios: DNA/RNA sequences, SMILES strings, protein sequences

Lesson 3 Code Example

```python
# DNA Sequence manipulation
dna = "ATGCGATCGATCG"
print(f"Length: {len(dna)}")
print(f"First 3 (codon): {dna[:3]}")   # ATG
print(f"Last codon: {dna[-3:]}")         # TCG

# Transcription: DNA -> RNA (T -> U)
rna = dna.replace("T", "U")
print(f"RNA: {rna}")   # AUGCGAUCGAUCG

# Count nucleotides
print(f"A: {dna.count('A')}, T: {dna.count('T')}")
print(f"G: {dna.count('G')}, C: {dna.count('C')}")

# SMILES analysis
smiles = "CC(=O)OC1=CC=CC=C1C(=O)O"
has_ring = any(c.isdigit() for c in smiles)
print(f"Has ring: {has_ring}")   # True
```

Lab 3: Strings

Objective: Manipulate SMILES and biological sequences.

Exercise 3.1 – DNA Transcription

Transcribe DNA to RNA:
Input: "ATGCGATCGATCG"
Replace all T with U
Expected: "AUGCGAUCGAUCG"

Exercise 3.2 – Reverse Complement (Rosalind REVC)

Generate reverse complement of DNA:
Input: "AAAACCCGGT"
Complement: A↔T, G↔C, then reverse
Expected: "ACCGGGTTTT"

Exercise 3.3 – SMILES Analysis

Analyze SMILES: "CC(=O)OC1=CC=CC=C1C(=O)O" (Aspirin)

- Find if it contains a ring (digits indicate ring closure)
- Count the number of carbons (C)
- Count oxygen atoms (O)

Exercise 3.4 – Nucleotide Count (Rosalind DNA)

Count nucleotides in: "AGCTTTTCATTCTGACTGCAACGGGCAATA"
Print: "A:X T:X G:X C:X"

Lesson 4: Learning Objectives

Learning Objectives:

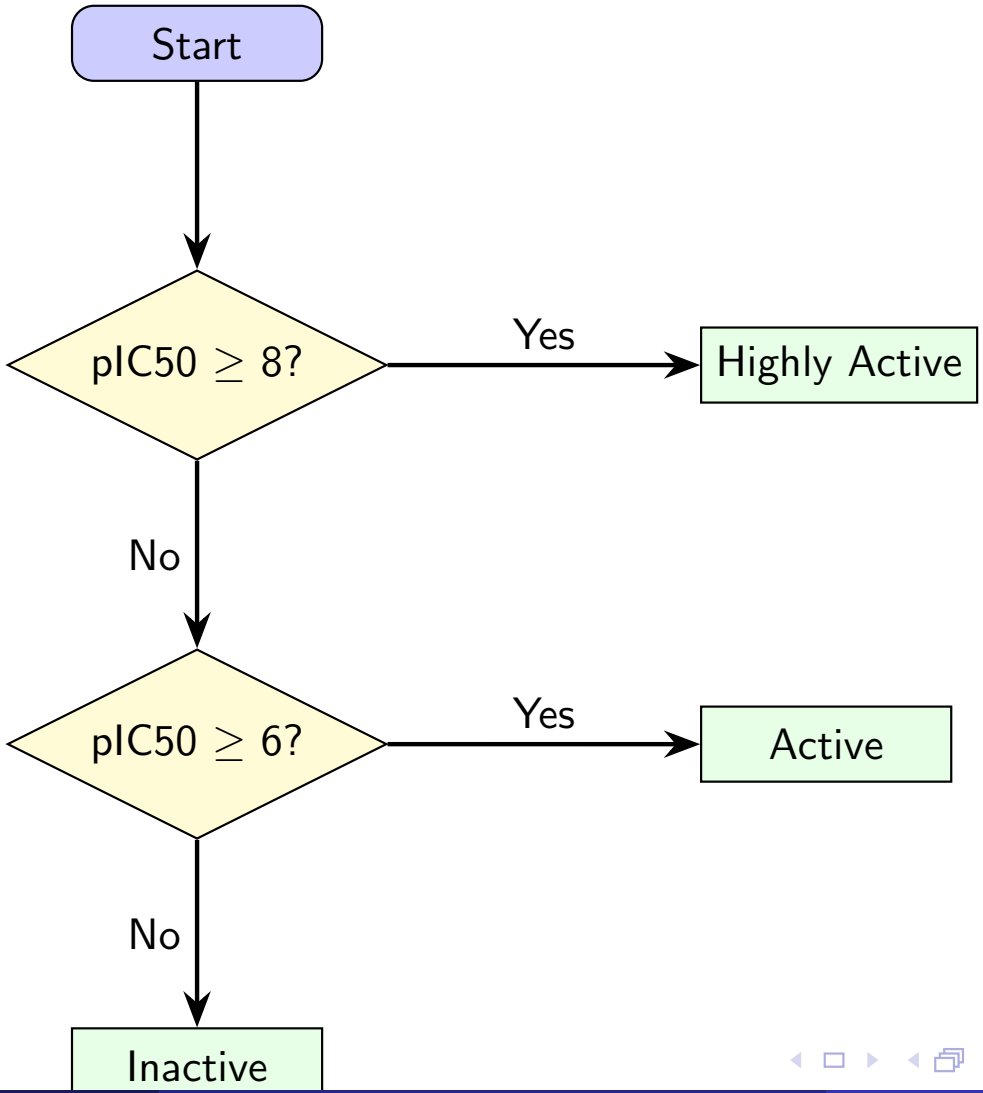
- Control program flow with if/elif/else statements
- Use match-case for pattern matching (Python 3.10+)
- Build nested conditional logic

Applications:

- Classify compounds by activity level
- Check drug-likeness (Lipinski violations)
- Identify start/stop codons in sequences

Lesson 4: Conditional Statements

`if/elif/else` for branching
`match-case` (Python 3.10+) for pattern matching



Lesson 4 Code Example

```python
# Classify compound activity by pIC50
pic50 = 7.5
if pic50 >= 8:
    activity = "Highly Active"
elif pic50 >= 6:
    activity = "Active"
elif pic50 >= 5:
    activity = "Moderate"
else:
    activity = "Inactive"
print(f"pIC50 {pic50}: {activity}")

# Codon identification (Python 3.10+)
codon = "ATG"
match codon:
    case "ATG":
        print("Start codon (Methionine)")
    case "TAA" | "TAG" | "TGA":
        print("Stop codon")
    case _:
        print("Coding codon")
```

Lab 4: Conditionals

Objective: Implement decision logic for compound classification.

Exercise 4.1 – Drug-Likeness Checker

Create a program that checks Lipinski Rule of Five:
Input: MW, LogP, HBD, HBA
Output: Number of violations (0–4)
Print "Drug-like" if violations ≤ 1, else "Non-drug-like"

Exercise 4.2 – Codon Identifier

Given a 3-letter codon, identify if it's:

- Start codon: "ATG"
- Stop codon: "TAA", "TAG", "TGA"
- Other: any other codon

Exercise 4.3 – Activity Classifier

Classify compound based on pIC50:

- pIC50 ≥ 8: "Highly Active"
- $7 \leq$ pIC50 < 8: "Active"
- $6 \leq$ pIC50 < 7: "Moderately Active"
- $5 \leq$ pIC50 < 6: "Weakly Active"
- pIC50 < 5: "Inactive"

Test with values: 8.5, 7.2, 6.5, 5.3, 4.1

Lesson 5: Learning Objectives

Learning Objectives:

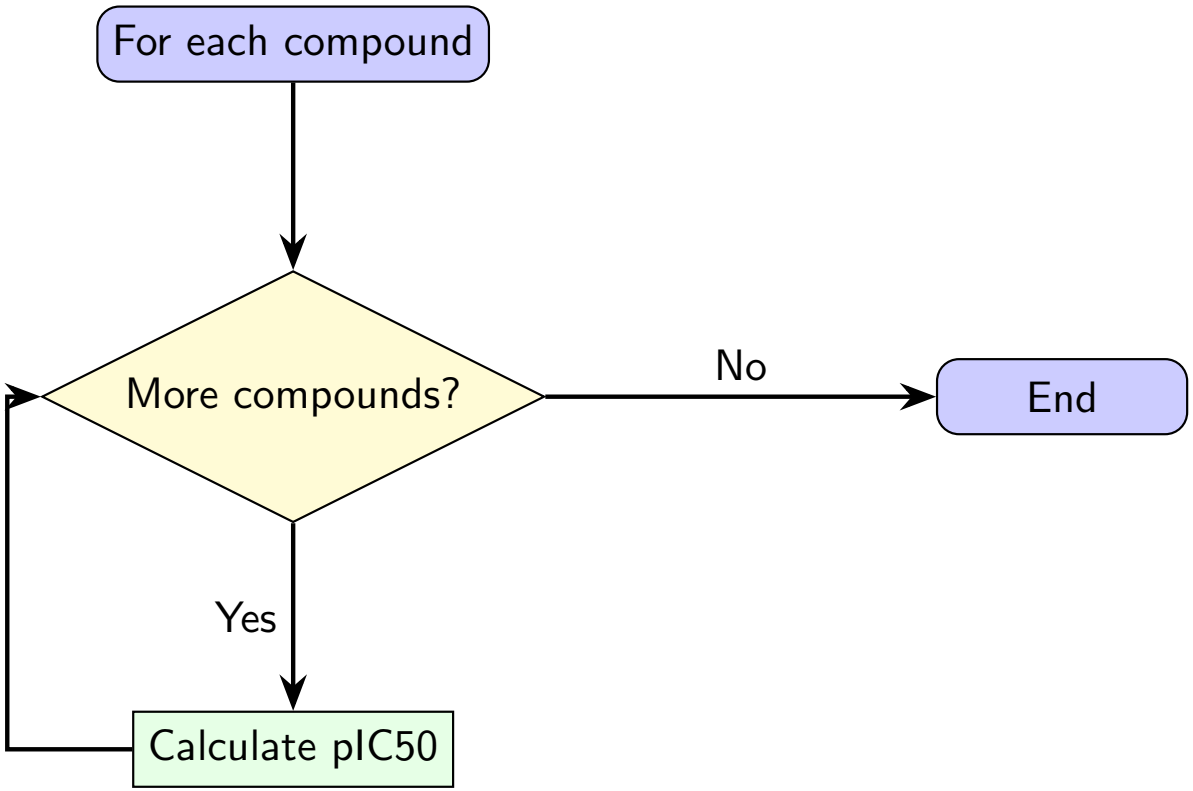
- Iterate over sequences using for loops
- Use while loops for conditional repetition
- Control loop flow with break and continue

Applications:

- Process compound libraries (batch MW calculation)
- Count nucleotide frequencies (Rosalind DNA)
- Screen compounds against activity thresholds

Lesson 5: Loops

`for` loop: iterate sequences/range
`while` loop: repeat until condition False
`break`/`continue`: control loop flow



Lesson 5 Code Example: For Loops

```python
import math

# Process compound library - calculate pIC50
ic50_values = [5.2, 120.0, 8.7, 2.1, 450.0]  # nM
for ic50 in ic50_values:
    pic50 = 9 - math.log10(ic50)
    print(f"IC50: {ic50:>6.1f} nM -> pIC50: {pic50:.2f}")

# Count nucleotides in DNA sequence
dna = "ATGCGATCGATCG"
counts = {"A": 0, "T": 0, "G": 0, "C": 0}
for nucleotide in dna:
    counts[nucleotide] += 1
print(f"Nucleotide counts: {counts}")

# Find active compounds (break/continue)
compounds = [("CPD1", 7.2), ("CPD2", 5.1), ("CPD3", 8.5)]
for name, pic50 in compounds:
    if pic50 < 6: continue  # skip inactive
    print(f"{name}: Active (pIC50={pic50})")
```

Lesson 5 Code Example: While Loops

```
# While loop: find first potent compound (pIC50 >= 7.5)
pic50_values = [5.2, 5.8, 6.1, 7.5, 8.2, 6.8]
i = 0
while i < len(pic50_values):
    if pic50_values[i] >= 7.5:
        print(f"First potent at index {i}: pIC50={
            pic50_values[i]}")
        break  # exit loop when found
    i += 1

# While loop with user input simulation
threshold = 6.0
compound_count = 0
max_compounds = 5
while compound_count < max_compounds:
    # Simulate reading compounds until reaching limit
    compound_count += 1
    print(f"Processing compound {compound_count}")

# While with sentinel value
sequence = ""
```

Lab 5: Loops

Objective: Process collections of compounds and sequences.

Exercise 5.1 – Batch IC50 Conversion

Convert list of IC50 values (nM) to pIC50:
IC50_list = [1.0, 10.0, 100.0, 1000.0, 10000.0]
Use a for loop to calculate and print each pIC50.

Exercise 5.2 – Nucleotide Counter (Rosalind DNA)

Count all nucleotides in a DNA sequence using a for loop:
seq = "AGCTTTTCATTCTGACTGCAACGGGCAATATGTCTCTGTGT"
Print counts for A, C, G, T separated by spaces.

Exercise 5.3 – Filter Active Compounds

Given pIC50 values: [5.2, 6.8, 7.3, 4.9, 8.1, 5.9, 6.2]
Use a for loop with continue to skip inactive (pIC50 < 6). Print only active compounds.

Exercise 5.4 – Find First Potent (While Loop)

Given pIC50 values: [5.2, 5.8, 6.1, 7.5, 8.2, 6.8]
Use a while loop with break to find the first "highly potent" compound (pIC50 ≥ 7.5).
Print its index and value.
Hint: Use index variable i, increment i += 1, check condition before break.

Exercise 5.5 – Read Until Stop Codon (While Loop)

Given codons: ["ATG", "CGA", "TCG", "GGC", "TAA", "AAA"]
Use a while loop to read codons and build a sequence string.
Stop when you encounter a stop codon ("TAA", "TAG", or "TGA").
Print the sequence built before the stop codon.

Exercise 5.6 – Compound Screening (While Loop)

Simulate screening compounds until finding 3 active ones:
Given: [4.5, 5.2, 6.8, 5.1, 7.3, 4.9, 8.1, 5.9]
Use while loop, count actives (pIC50 $\geq$ 6), stop when count reaches 3.
Print how many compounds were screened.

Lesson 6: Learning Objectives

Learning Objectives:

- Define reusable functions with parameters
- Use return statements to output results
- Apply *args and **kwargs for flexible inputs

Applications:

- Create IC50 $\rightarrow$ pIC50 converters
- Build Lipinski property calculators
- Implement sequence analysis functions (GC, REVC)

Lesson 6: Functions

`def name(params):` define function
`return` for return value

Special Parameters:

- `*args` – accepts any number of **positional** arguments as a tuple
- `**kwargs` – accepts any number of **keyword** arguments as a dictionary

Lesson 6 Code Example: Basic Functions

```python
import math

# Function: IC50 to pIC50 conversion
def ic50_to_pic50(ic50_nm):
    """Convert IC50 (nM) to pIC50."""
    return -math.log10(ic50_nm * 1e-9)

# Function with validation (returns tuple)
def calculate_mw(smiles):
    """Calculate MW from SMILES using RDKit."""
    from rdkit import Chem
    from rdkit.Chem import Descriptors
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return None, "Invalid SMILES"
    return Descriptors.MolWt(mol), "Success"

# Usage
print(ic50_to_pic50(10))  # 8.0
mw, status = calculate_mw("CCO")
print(f"MW: {mw:.2f}, Status: {status}")
```

Lesson 6 Code Example: *args and **kwargs

```python
# *args - accept variable number of positional arguments
def average_activity(*pic50_values):
    """Calculate average pIC50 from multiple values."""
    return sum(pic50_values) / len(pic50_values)

# **kwargs - accept variable number of keyword arguments
def print_compound(**props):
    """Print compound properties as key-value pairs."""
    for key, value in props.items():
        print(f"{key}: {value}")

# Usage examples
avg = average_activity(5.2, 6.8, 7.3, 8.1)
print(f"Average pIC50: {avg:.2f}")

print_compound(name="Aspirin", MW=180.16, pIC50=5.2)
```

Lab 6: Functions

Objective: Create reusable molecular utility functions.

Exercise 6.1 – pIC50 Converter Function

Create function: `ic50_to_pic50(ic50_nm)`
Input: IC50 in nanomolar. Output: pIC50 value.
Test with: 10, 100, 1000 nM

Exercise 6.2 – GC Content Function

Create function: `gc_content(sequence)`
Input: DNA sequence string. Output: GC percentage (float).
Test with: "AGCTATAG", "GCGCGCGC", "ATATAT"

Exercise 6.3 – Lipinski Calculator

Create function: `check_lipinski(mw, logp, hbd, hba)`
Returns tuple: (passes: bool, violations: int). Test with multiple compound property sets.

Exercise 6.4 – Reverse Complement Function

Create function: `reverse_complement(dna)`
Input: DNA sequence. Output: Reverse complement sequence.
Test with Rosalind REVC sample: "AAAACCCGGT" $\rightarrow$ "ACCGGGTTTT"

Lesson 6B: Learning Objectives

Learning Objectives:

- Handle runtime errors with try/except blocks
- Use else and finally for cleanup operations
- Raise custom exceptions for validation

Applications:

- Handle invalid SMILES parsing gracefully
- Manage missing data in compound datasets
- Validate FASTA file formats

Lesson 6B: Error Handling (try/except)

Concept: Handle runtime errors gracefully

Keywords: `try`, `except`, `else`, `finally`, `raise`

Common Exceptions:

- `ValueError` – invalid value conversion
- `TypeError` – wrong type operation
- `ZeroDivisionError` – division by zero
- `FileNotFoundError` – file doesn't exist
- `KeyError` – dict key not found

Lesson 6B Code Example

```
# Basic try/except
try:
    num = int(input("Enter number: "))
    result = 10 / num
except ValueError:
    print("Invalid input!")
except ZeroDivisionError:
    print("Cannot divide by zero!")
else:
    print(f"Result: {result}")
finally:
    print("Execution complete")

# Raising exceptions
def divide(a, b):
    if b == 0:
        raise ValueError("Divisor cannot be zero")
    return a / b
```

Lab 6B: Error Handling

Objective: Build robust code that handles invalid inputs.

Exercise 6B.1 – Safe IC50 Conversion

Modify `ic50_to_pic50()` to handle: Negative IC50 values (raise ValueError), Zero IC50 (raise ValueError), Non-numeric input (catch TypeError).
Return None on error and print helpful message.

Exercise 6B.2 – SMILES Validator

Create function that validates SMILES:
Use RDKit: `Chem.MolFromSmiles(smiles)`
If returns None, raise ValueError with message.
Handle with try/except and return valid/invalid status.

Lesson 7: Learning Objectives

Learning Objectives:

- Create and modify lists using built-in methods
- Access elements via indexing and slicing
- Perform common list operations (append, remove, sort)

Applications:

- Store SMILES strings for compound libraries
- Manage pIC50 activity measurements
- Build queues for batch processing

Lesson 7: Python Lists

Lists store ordered sequences.

Methods: `append`, `extend`, `insert`, `remove`, `pop`, `clear`, `index`, `count`, `copy`

Scenarios: SMILES list, pIC50 values, compound IDs

Lesson 7 Code Example

```
smiles_list = ["CCO", "CC(=O)O", "c1ccccc1"]
smiles_list.append("CCN")
smiles_list.insert(1, "CC")
smiles_list.remove("CCO")
print(smiles_list[0], smiles_list[-1])

# pIC50 statistics
pic50_values = [5.2, 6.8, 7.3, 4.9, 8.1]
print(f"Min: {min(pic50_values)}")
print(f"Max: {max(pic50_values)}")
print(f"Sorted: {sorted(pic50_values)}")
```

Lab 7: Lists

Objective: Manage compound libraries using lists.

Exercise 7.1 – Compound Library

Create a list of SMILES strings for 5 common drugs.
Perform operations:

- Add a new compound
- Remove a compound by value
- Insert at specific position
- Print first and last compounds

Exercise 7.2 – pIC50 Statistics

Given: [5.2, 6.8, 7.3, 4.9, 8.1, 5.9, 6.2, 7.8]
Calculate: min, max, sorted list, count of actives ($\geq$6)

Lesson 7B: Learning Objectives

Learning Objectives:

- Use tuples for immutable data records
- Apply sets for unique element collections
- Perform set operations (union, intersection, difference)

Applications:

- Store compound records (name, SMILES, pIC50)
- Find unique molecular scaffolds
- Compare compound libraries

Lesson 7B Code Example

```
# Tuples - immutable (compound data)
compound = ("Aspirin", "CC(=O)OC1=CC=CC=C1C(=O)O", 180.16)
name, smiles, mw = compound  # unpacking

# Sets - unique scaffolds
scaffolds = {"benzene", "pyridine", "benzene"}  # 2 unique
scaffolds.add("furan")

# Set operations for compound comparison
lib_A = {"CMP001", "CMP002", "CMP003"}
lib_B = {"CMP002", "CMP003", "CMP004"}
print(lib_A | lib_B)  # union: all compounds
print(lib_A & lib_B)  # intersection: common
print(lib_A - lib_B)  # unique to lib_A
```

Lab 7B: Tuples & Sets

Exercise 7B.1 – Compound Records

Create tuples for 3 compounds: (name, SMILES, pIC50)
Unpack each tuple and print formatted output.
Try to modify a tuple value – observe the error.

Exercise 7B.2 – Library Comparison

Library A: {"CMP001", "CMP002", "CMP003", "CMP004"}
Library B: {"CMP003", "CMP004", "CMP005", "CMP006"}
Find:

- All unique compounds (union)
- Common compounds (intersection)
- Compounds only in A / only in B

Lesson 8: Learning Objectives

Learning Objectives:

- Write concise list comprehensions
- Apply map() and filter() for transformations
- Combine functional programming techniques

Applications:

- Filter active compounds (pIC50 > 6)
- Batch convert IC50 to pIC50 values
- Extract drug-like compounds

Lesson 8 Code Example

```
import math
pic50_values = [5.2, 6.8, 7.3, 4.9, 8.1]

# List comprehension: filter active compounds
actives = [p for p in pic50_values if p >= 6.0]

# Lambda + map: convert pIC50 to IC50 (nM)
ic50_nm = list(map(lambda p: 10**(9-p), pic50_values))

# Filter: highly potent (pIC50 > 7)
potent = list(filter(lambda p: p > 7, pic50_values))

print(f"Active: {actives}")
print(f"Potent: {potent}")
```

Lab 8: List Comprehensions

Exercise 8.1 – Filter Active Compounds

Given pIC50 = [5.2, 6.8, 7.3, 4.9, 8.1, 5.9]
Use list comprehension to filter pIC50 $\geq$ 6.0

Exercise 8.2 – Batch Conversion

Convert IC50 list [10, 100, 1000] to pIC50 using:
a) List comprehension b) map() with lambda

Exercise 8.3 – Conditional Comprehension

Create list of tuples: [(pIC50, "Active") if pIC50 $\geq$ 6 else (pIC50, "Inactive")]
for values [5.2, 6.8, 7.3, 4.9, 8.1]

Lesson 9: Learning Objectives

Learning Objectives:

- Create and manipulate key-value dictionaries
- Access, update, and iterate over dict items
- Use dict comprehensions for transformations

Applications:

- Build compound databases (name $\rightarrow$ properties)
- Create codon translation tables
- Store molecular descriptor lookups

Lesson 9 Code Example

```
compound_db = {
    "Aspirin": {"SMILES": "CC(=O)OC1=CC=CC=C1C(=O)O", "pIC50
        ": 5.2},
    "Caffeine": {"SMILES": "CN1C=NC2=C1C(=O)N(C(=O)N2C)C", "
        pIC50": 4.8}
}

# Add new compound
compound_db["Ibuprofen"] = {
    "SMILES": "CC(C)CC1=CC=C(C=C1)C(C)C(=O)O",
    "pIC50": 6.1
}

# Filter actives (dict comprehension)
actives = {k: v for k, v in compound_db.items() if v["pIC50"
    ] >= 5.0}

print(list(actives.keys()))
```

Lab 9: Dictionaries

Exercise 9.1 – Compound Database

Create a dict of compounds with nested properties:
Key: compound name, Value: dict with SMILES, MW, pIC50, is_active

Exercise 9.2 – Codon Table (Rosalind)

Create a dict mapping codons to amino acids:
"ATG" $\rightarrow$ "M", "TGG" $\rightarrow$ "W", "TAA" $\rightarrow$ "Stop"
Use to translate a short sequence.

Exercise 9.3 – Dict Comprehension

Filter the compound database to only active compounds
using dict comprehension: {k:v for k,v in ... if ...}

Lesson 10: Learning Objectives

Learning Objectives:

- Read and write text files using open()
- Use context managers (with) for safe file handling
- Parse structured file formats (CSV, FASTA)

Applications:

- Read/write compound CSV files
- Parse FASTA sequence files
- Export filtered results

Lesson 10 Code Example

```python
# Write compound data to CSV
with open("compounds.csv", "w") as f:
    f.write("name,smiles,pIC50\n")
    f.write("Aspirin,CC(=O)OC1=CC=CC=C1C(=O)O,5.2\n")

# Read FASTA file
with open("sequence.fasta", "r") as f:
    header = f.readline().strip()  # >sequence_id
    sequence = ""
    for line in f:
        sequence += line.strip()

print(f"Header: {header}")
print(f"Sequence length: {len(sequence)}")
```

Lab 10: File Handling

Exercise 10.1 – Write Compound CSV

Create a CSV file with columns: name, SMILES, pIC50
Write data for 5 compounds using `with open()`.

Exercise 10.2 – Read FASTA

Create a simple FASTA parser: Read file, extract header (lines starting with $>$), concatenate sequence lines, return dict: {header: sequence}

Exercise 10.3 – Filter and Export

Read compound CSV, filter active compounds (pIC50 $\geq$ 6)
Write filtered results to new file "actives.csv"

Lesson 11: Learning Objectives

Learning Objectives:

- Create and manipulate NumPy arrays
- Perform vectorized mathematical operations
- Use boolean indexing for data filtering

Applications:

- Store molecular descriptor matrices
- Normalize and scale feature data
- Compute statistics on activity arrays

Lesson 11 Code Example

```
import numpy as np

# Array creation
arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
zeros = np.zeros((2, 3))
seq = np.arange(0, 10, 2)  # [0, 2, 4, 6, 8]

# Properties
print(arr.shape, arr.dtype, arr.ndim)

# Operations
print(arr * 2)             # element-wise multiply
print(arr.sum(axis=0))     # sum per column
print(arr.mean(axis=1))    # mean per row

# Boolean indexing
mask = arr > 5
print(arr[mask])  # [6, 7, 8, 9]
```

Lab 11: NumPy

Exercise 11.1 – Descriptor Matrix

Create a 2D array of molecular descriptors:
Rows = compounds, Columns = [MW, LogP, HBD, HBA]
Calculate mean and std for each descriptor (column).

Exercise 11.2 – Normalization

Normalize descriptor values to 0-1 range: (x - min) / (max - min)
Use vectorized operations (no loops).

Exercise 11.3 – Boolean Filtering

Filter compounds where MW $<$ 500 AND LogP $<$ 5
Use boolean indexing.

Lesson 11B: Learning Objectives

Learning Objectives:

- Create Series and DataFrame structures
- Filter, group, and aggregate tabular data
- Read/write data from CSV files

Applications:

- Manage compound libraries with properties
- Analyze bioactivity data
- Merge descriptor and activity datasets

Lesson 11B Code Example

```
import pandas as pd

df = pd.DataFrame({
    'Name': ['Aspirin', 'Ibuprofen', 'Caffeine', 'Drug_X'],
    'pIC50': [5.2, 6.1, 4.8, 7.5],
    'MW': [180.16, 206.28, 194.19, 320.5]
})

# Selection & Filtering
actives = df[df['pIC50'] > 6.0]
drug_like = df[df['MW'] < 500]

# Aggregation
print(df['pIC50'].mean())
print(df.describe())

# GroupBy
df['Class'] = df['pIC50'].apply(lambda x: 'Active' if x >= 6
    else 'Inactive')
grouped = df.groupby('Class')['MW'].mean()
```

Lab 11B: Pandas

Exercise 11B.1 – Create Compound DataFrame

Create DataFrame with: Name, SMILES, MW, LogP, pIC50
Add 5+ compounds. Add column for activity class.

Exercise 11B.2 – Data Analysis

Calculate: mean pIC50, count by activity class
Filter drug-like compounds (MW $<$ 500, LogP $<$ 5). Sort by pIC50 descending.

Exercise 11B.3 – GroupBy Analysis

Group compounds by activity class
Calculate mean MW and LogP per group. Export results to CSV.

Lesson 12: Learning Objectives

Learning Objectives:

- Parse and generate JSON data
- Write regex patterns for text matching
- Extract and validate patterns in sequences

Applications:

- Query ChEMBL/PubChem REST APIs
- Find restriction sites in DNA sequences
- Validate SMILES and sequence formats

Lesson 12 Code Example

```
import json
import re

# JSON - PubChem-like data
data = '{"name": "Aspirin", "CID": 2244, "MW": 180.16}'
compound = json.loads(data)
print(compound["name"])

# Regex - find DNA motifs
seq = "ATGCGATCGATCG"
matches = re.findall(r"GATC", seq)  # restriction site
print(f"GATC sites: {len(matches)}")

# Find EcoRI sites
seq2 = "ATGAATTCGCGAATTCTA"
for match in re.finditer(r"GAATTC", seq2):
    print(f"EcoRI site at position {match.start()}")
```

Lab 12: JSON & Regex

Exercise 12.1 – Parse PubChem JSON

Parse JSON compound data: `{"CID": 2244, "name": "Aspirin", "MW": 180.16}`
Extract and print each field.

Exercise 12.2 – Find Restriction Sites

Use regex to find all "GAATTC" (EcoRI site) in: "ATGAATTCGCGAATTCTA"
Print positions of each match.

Exercise 12.3 – SMILES Validation

Use regex to check if SMILES contains aromatic ring (lowercase c, n), ring closure (digits), or double bond (=).

Rosalind Challenges

Complete these Rosalind.info problems:

1. **DNA** – Counting DNA Nucleotides
2. **RNA** – Transcribing DNA into RNA
3. **REVC** – Complementing a Strand of DNA
4. **GC** – Computing GC Content
5. **HAMM** – Counting Point Mutations
6. **PROT** – Translating RNA into Protein
7. **SUBS** – Finding a Motif in DNA
8. **CONS** – Consensus and Profile

Submission: Upload your solutions to GitHub with clear function documentation and test cases.

Mini-Project: QSAR Data Prep Pipeline

Objective: Build a complete data preparation pipeline. **Tasks:**

1. Read compound CSV with SMILES and IC50 values
2. Validate all SMILES using RDKit
3. Convert IC50 (nM) to pIC50
4. Calculate Lipinski descriptors (MW, LogP, HBD, HBA)
5. Add activity class column (Active/Inactive)
6. Filter drug-like compounds
7. Export clean dataset to CSV
8. Generate summary statistics

Deliverables: Python script, output CSV, summary report

Course Summary

Section 1: Python Basics (Lessons 1–6B)

- Variables, Data Types, Operators
- Strings (SMILES, DNA sequences)
- Conditionals and Loops
- Functions and Error Handling

Section 2: Collections (Lessons 7–12)

- Lists, Tuples, Sets, Dictionaries
- List Comprehensions, Lambda Functions
- File Handling (CSV, FASTA)
- NumPy, Pandas
- JSON, Regex

Practice: Rosalind challenges + QSAR mini-project

Next Steps

Practice Resources:

- Rosalind.info for bioinformatics problems
- ChEMBL/PubChem for real compound data
- RDKit tutorials for cheminformatics

Topics to Explore Next:

- Object-Oriented Programming
- Machine Learning (scikit-learn)
- QSAR/QSPR modeling
- Molecular visualization
- Deep learning with molecular graphs

Good luck with your exercises!