

Study Delight Publications

PYTHON

Simplified

Build
Stronger
Python
Skills

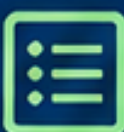
Learn
Practice
Create
Grow

PART 2

Core Skills – Strings & Formatting



Strings
& Formatting



Collections



Functions



Modules
& Tools



From
Basics to
Real-World
Programs

Practice
Code
Build
Real Skills



Clear
Explanations



Hands-on
Examples



Practical
Exercises



Real-World
Applications



Build
Confidence

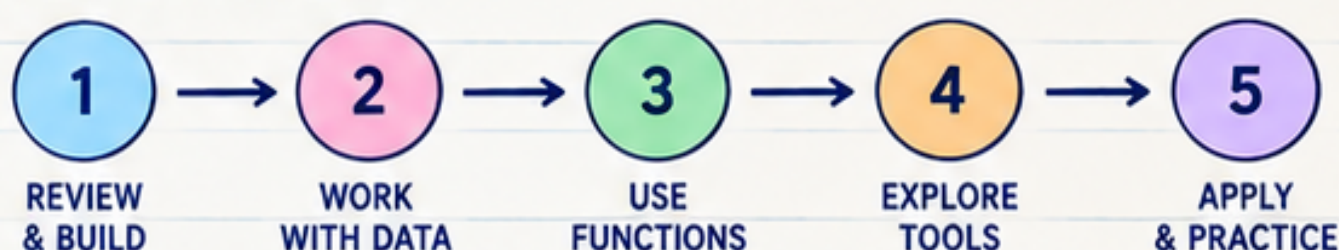
pythonforaiml.com

PYTHON PART 2

Core Skills – Strings & Formatting

Take your Python skills to the next level. Work with strings, collections, functions and more to write clean, useful and real-world programs.

30-PAGE ROADMAP



01 Part 2 Overview

02 Strings and Formatting

03 String Methods

04 String Searching

05 String Slicing

06 String Splitting & Joining

07 String Practice Problems

08 Lists in Detail

09 List Methods

10 List Comprehensions

11 Tuples

12 Sets

13 Dictionaries

14 Dictionary Methods

15 Nested Collections

17 Functions

17 Function Parameters

18 Return Values

19 Scope: Local vs Global

20 Lambda Functions

21 Modules and Imports

22 Useful Standard Modules

23 Working with Files

24 Error Handling (Try-Except)

25 Mini Project: Student Manager

26 Best Practices

27 Common Mistakes

28 Quick Revision

29 Practice Exercises

30 What's Next? (Part 3 Preview)

LEARN • PRACTICE • BUILD • GROW

Use this index as your quick navigation map while studying.

Small
Steps
Big Progress
! 😊

Part 2: Python Core Skills — Strings & Formatting

Explanation

Part 2 develops the skills that turn basic Python syntax into useful programs. We go deeper into strings, collections, comprehensions, functions, functional tools and code organisation.

An f-string lets you place variables and expressions directly inside a string. It is one of the clearest ways to create dynamic messages because the data and its presentation appear together.

Example

```
name="Asha"  
age=21  
print(f"Name: {name}")  
print(f"Age: {age}")
```

Output

```
Name: Asha  
Age: 21
```

Remember

The f before the opening quote tells Python to evaluate expressions inside braces { }.

Practice

Create a message showing your name, course and current learning goal.

Study Loop

- Predict the output before running the example.
- Change one value and observe what happens.
- Explain the result in your own words.
- Write a small variation without copying the example.



2. String Formatting

Explanation

Formatting is useful when numbers need a consistent appearance. A format specification after a colon controls how a value is displayed.

Example

```
price=125.6789
print(f"₹{price:.2f}")
print(f"{42:05d}")
```

Output

```
₹125.68
00042
```

Formatting Drill

.2f means two decimal places. 05d means an integer displayed in a width of five with leading zeroes.

Practice

Format a percentage with exactly one decimal place.

Study Loop

- Predict the output before running the example.
- Change one value and observe what happens.
- Explain the result in your own words.
- Write a small variation without copying the example.



3. String Searching

Explanation

Text-processing programs often need to ask questions such as: does this text contain a word, or does a filename end with a particular extension? Python has simple methods for these checks.

Example

```
email="student@example.com"
print("@" in email)
print(email.startswith("student"))
print(email.endswith(".com"))
print(email.find("@"))
```

Output

```
True
True
True
7
```

Quick Check

`in` checks membership. `startswith()` and `endswith()` check boundaries. `find()` returns the first index or `-1`.

Practice

Check whether a filename ends with `.csv` and whether a sentence contains `Python`.

Study Loop

- Predict the output before running the example.
- Change one value and observe what happens.
- Explain the result in your own words.
- Write a small variation without copying the example.



4. Lists: Mutation

Explanation

Lists are mutable. That means an existing list can be changed after it is created. This is useful for shopping carts, task queues and collections that grow over time.

Example

```
tasks=["read","code"]
tasks.append("practice")
tasks.insert(1,"plan")
print(tasks)
```

Output

```
['read', 'plan', 'code', 'practice']
```

List Practice

append() adds at the end. insert(index,value) places an item at a position. The original list is changed.

Practice

Create a list of three subjects, insert one at position 1, then append another.

Study Loop

- Predict the output before running the example.
- Change one value and observe what happens.
- Explain the result in your own words.
- Write a small variation without copying the example.



5. Lists: pop() and remove()

Explanation

`remove()` searches for a value and removes its first matching occurrence. `pop()` removes an item by position and returns the removed value, which is useful when the removed item matters.

Example

```
items=["a","b","c"]
removed=items.pop(1)
items.remove("a")
print(removed)
print(items)
```

Output

```
b
['c']
```

Compare

Use `remove(value)` when you know the value. Use `pop(index)` when you know the position and want the removed item.

Practice

Explain why `pop()` is useful for processing the last item of a list.

Study Loop

- Predict the output before running the example.
- Change one value and observe what happens.
- Explain the result in your own words.
- Write a small variation without copying the example.



6. Sorting Lists

Explanation

Sorting is common when data must be presented in an ordered way. `sorted()` creates a new sorted list, while `sort()` changes the original list.

Example

```
scores=[72,91,65,88]
print(sorted(scores))
print(sorted(scores,reverse=True))
print(scores)
```

Output

```
[65, 72, 88, 91]
[91, 88, 72, 65]
[72, 91, 65, 88]
```

Important

Use `sorted()` when you want to preserve the original order. Use `sort()` when changing the list is intentional.

Practice

Sort a list of names alphabetically and then in reverse order.

Study Loop

- Predict the output before running the example.
- Change one value and observe what happens.
- Explain the result in your own words.
- Write a small variation without copying the example.



7. Nested Lists

Explanation

A list can contain other lists. This is useful for small tables, grids and grouped data. Read nested indexes from left to right: first choose the inner list, then choose an item inside it.

Example

```
marks=[[80,75,90],[65,88,70]]  
print(marks[0])  
print(marks[1][2])
```

Output

```
[80, 75, 90]  
70
```

Think About It

marks[1] selects the second row. [2] then selects the third value in that row.

Practice

Build a 3*3 number grid and print its centre value.

Study Loop

- Predict the output before running the example.
- Change one value and observe what happens.
- Explain the result in your own words.
- Write a small variation without copying the example.



8. Tuples & Unpacking

Explanation

Tuples are ordered collections that are commonly used for fixed groups of related values. Unpacking lets you assign their elements to meaningful names in one statement.

Example

```
student=("Asha",21,"Python")
name,age,course=student
print(name)
print(course)
```

Output

```
Asha
Python
```

Quick Check

Unpacking makes the structure explicit. The number of target variables normally needs to match the number of values.

Practice

Create a tuple for a point (x,y) and unpack it into x and y.

Study Loop

- Predict the output before running the example.
- Change one value and observe what happens.
- Explain the result in your own words.
- Write a small variation without copying the example.



9. Sets: Unique Data

Explanation

Sets are designed for unique values and membership operations. When duplicate values are not meaningful, a set can remove them automatically.

Example

```
nums={1,2,2,3,3,3}
print(nums)
nums.add(4)
print(2 in nums)
```

Output

```
{1, 2, 3}
True
```

Set Thinking

Sets are not indexed like lists. Do not rely on their displayed order.

Practice

Create a set from a list containing duplicate city names.

Study Loop

- Predict the output before running the example.
- Change one value and observe what happens.
- Explain the result in your own words.
- Write a small variation without copying the example.



10. Set Operations

Explanation

Set operations express useful comparisons between groups. Union combines values, intersection finds common values, and difference finds values present in the first set but not the second.

Example

```
a={"Python", "SQL", "Git"}
b={"Python", "AI", "Pandas"}
print(a|b)
print(a&b)
print(a-b)
```

Output

```
{'Python', 'SQL', 'Git', 'AI', 'Pandas'}
{'Python'}
{'SQL', 'Git'}
```

Practice

The exact display order of a set is not important. Focus on which values belong to the result.

Practice

Create two skill sets and find their common skills.

Study Loop

- Predict the output before running the example.
- Change one value and observe what happens.
- Explain the result in your own words.
- Write a small variation without copying the example.



11. Dictionaries: get()

Explanation

A dictionary stores key-value pairs. Bracket access is appropriate when a key is expected to exist. `get()` is useful when missing data is a normal possibility.

Example

```
user={"name": "Asha", "city": "Pune"}  
print(user["name"])  
print(user.get("age"))  
print(user.get("age", "Unknown"))
```

Output

```
Asha  
None  
Unknown
```

Decision Guide

`get(key, default)` avoids an immediate `KeyError` and gives you a controlled fallback.

Practice

Read a missing product price with `get()` and provide a default message.

Study Loop

- Predict the output before running the example.
- Change one value and observe what happens.
- Explain the result in your own words.
- Write a small variation without copying the example.



12. Dictionary Iteration

Explanation

keys(), values() and items() provide different ways to inspect dictionary data. When a loop needs both the key and its value, items() is usually the clearest choice.

Example

```
student={"name":"Asha","age":21}  
for key,value in student.items():  
    print(key,"=>",value)
```

Output

```
name => Asha  
age => 21
```

Loop Pattern

items() avoids a second lookup and communicates that the loop needs both pieces of information.

Practice

Loop through a product dictionary and print each field and value.

Study Loop

- Predict the output before running the example.
- Change one value and observe what happens.
- Explain the result in your own words.
- Write a small variation without copying the example.



13. Dictionary Merging

Explanation

Dictionaries can be combined. With the `|` operator, keys from the right-hand dictionary replace values for duplicate keys in the left-hand dictionary.

Example

```
base={"name": "Asha", "age": 21}
extra={"age": 22, "city": "Pune"}
print(base|extra)
```

Output

```
{'name': 'Asha', 'age': 22, 'city': 'Pune'}
```

Important

When keys overlap, the right-hand value wins. This is useful for simple configuration or record updates.

Practice

Merge two dictionaries and predict the result for one shared key.

Study Loop

- Predict the output before running the example.
- Change one value and observe what happens.
- Explain the result in your own words.
- Write a small variation without copying the example.



14. List Comprehensions

Explanation

A list comprehension creates a new list by describing the transformation in a compact form. It is often easier to read than a multi-line loop when the operation is simple.

Example

```
nums=[1,2,3,4,5]
squares=[n*n for n in nums]
even=[n for n in nums if n%2==0]
print(squares)
print(even)
```

Output

```
[1, 4, 9, 16, 25]
[2, 4]
```

Recipe

Read it as: for each item, calculate this expression, optionally keeping only items that pass the condition.

Practice

Create a list containing the lengths of three words.

Study Loop

- Predict the output before running the example.
- Change one value and observe what happens.
- Explain the result in your own words.
- Write a small variation without copying the example.



15. Set & Dict Comprehensions

Explanation

The comprehension pattern also works with sets and dictionaries. The container changes, but the idea remains: iterate, calculate, and optionally filter.

Example

```
nums=[1,2,2,3]
print({n*n for n in nums})
print({n:n*n for n in nums})
```

Output

```
{1, 4, 9}
{1: 1, 2: 4, 3: 9}
```

Three Containers

A set removes duplicate results. A dictionary comprehension creates key-value pairs.

Practice

Create a dictionary mapping each number from 1 to 5 to its square.

Study Loop

- Predict the output before running the example.
- Change one value and observe what happens.
- Explain the result in your own words.
- Write a small variation without copying the example.



16. Nested Comprehensions

Explanation

A comprehension may contain more than one for clause. This can represent combinations or a small Cartesian product, but readability should always be checked.

Example

```
pairs=[(x,y) for x in [1,2] for y in [10,20]]  
print(pairs)
```

Output

```
[(1, 10), (1, 20), (2, 10), (2, 20)]
```

Read Slowly

Think of it as the outer loop choosing x and, for every x , the inner loop choosing y .

Practice

Create coordinate pairs for x in $[0,1]$ and y in $[0,1,2]$.

Study Loop

- Predict the output before running the example.
- Change one value and observe what happens.
- Explain the result in your own words.
- Write a small variation without copying the example.



17. Functions: Defaults

Explanation

Default parameters provide a value when the caller does not supply one. They are useful for optional behaviour while keeping the function call simple.

Example

```
def greet(name,message="Welcome"):  
    return f"{message}, {name}!"  
print(greet("Asha"))  
print(greet("Ravi","Hello"))
```

Output

```
Welcome, Asha!  
Hello, Ravi!
```

Function Design

The default is used only when that argument is omitted. Supplying a value overrides it.

Practice

Write a function that formats a message with a default language or greeting.

Study Loop

- Predict the output before running the example.
- Change one value and observe what happens.
- Explain the result in your own words.
- Write a small variation without copying the example.



18. Keyword Arguments

Explanation

Keyword arguments pass values by parameter name rather than position. They can make calls easier to read and can be supplied in a different order.

Example

```
def profile(name,age,city):  
    return f"{name} | {age} | {city}"  
print(profile(city="Pune",name="Asha",age=21))
```

Output

Asha | 21 | Pune

Call Styles

Positional arguments depend on order. Keyword arguments depend on parameter names.

Practice

Call a function with three keyword arguments in reverse order.

Study Loop

- Predict the output before running the example.
- Change one value and observe what happens.
- Explain the result in your own words.
- Write a small variation without copying the example.



19. *args

Explanation

When a function should accept any number of positional arguments, *args collects the extra values into a tuple.

Example

```
def total(*args):  
    return sum(args)  
print(total(10,20))  
print(total(1,2,3,4))
```

Output

```
30  
10
```

args Checklist

The name args is conventional; the star is the important part. Inside the function, args behaves like a tuple.

Practice

Write average(*numbers) and return the average of all supplied numbers.

Study Loop

- Predict the output before running the example.
- Change one value and observe what happens.
- Explain the result in your own words.
- Write a small variation without copying the example.



20. **kwargs

Explanation

****kwargs** collects extra keyword arguments into a dictionary. It is useful for flexible options and configuration-like interfaces.

Example

```
def show(**kwargs):  
    for key,value in kwargs.items():  
        print(key,"=",value)  
show(name="Asha",city="Pune")
```

Output

```
name = Asha  
city = Pune
```

kwargs Checklist

title receives the first positional value; items receives the remaining positional values; info receives named values.

Practice

Create a function that accepts any number of named student details.

Study Loop

- Predict the output before running the example.
- Change one value and observe what happens.
- Explain the result in your own words.
- Write a small variation without copying the example.



21. Combining Arguments

Explanation

A function can combine normal parameters, `*args` and `**kwargs`. The order in the definition matters because Python needs to know which values belong to which category.

Example

```
def report(title,*items,**info):  
    print(title)  
    print(items)  
    print(info)  
report("Student","Python","AI",city="Pune")
```

Output

```
Student  
( 'Python', 'AI' )  
{ 'city': 'Pune' }
```

API Thinking

title receives the first positional value; items receives the remaining positional values; info receives named values.

Practice

Explain which argument goes into each parameter in the example.

Study Loop

- Predict the output before running the example.
- Change one value and observe what happens.
- Explain the result in your own words.
- Write a small variation without copying the example.



22. Scope

Explanation

Scope determines where a variable name can be accessed. A variable created inside a function is normally local to that function, even when the same name exists outside it.

Example

```
x=10
def demo():
    x=20
    print("inside",x)
demo()
print("outside",x)
```

Output

```
inside 20
outside 10
```

Scope Rule

Passing values into functions and returning results usually creates clearer dependencies than relying heavily on global state.

Practice

Write a function with a local variable and compare it with a global variable of the same name.

Study Loop

- Predict the output before running the example.
- Change one value and observe what happens.
- Explain the result in your own words.
- Write a small variation without copying the example.



23. return vs print

Explanation

print() displays information for a person. return sends a result back to the caller. Returned values can be stored, transformed, tested or passed to another function.

Example

```
def add(a,b):  
    return a+b  
result=add(5,7)  
print(result*2)
```

Output

24

Golden Rule

Returning makes a function reusable. A function that only prints its result is harder to combine with other calculations.

Practice

Convert a printing function into one that returns its result.

Study Loop

- Predict the output before running the example.
- Change one value and observe what happens.
- Explain the result in your own words.
- Write a small variation without copying the example.



24. Lambda Functions

Explanation

A lambda is a small anonymous function written as an expression. It is most useful when a short function is needed temporarily, such as a sorting key.

Example

```
nums=[3,1,4,2]
print(sorted(nums,key=lambda n:n*n))
```

Output

```
[1, 2, 3, 4]
```

Lambda Rule

The lambda here computes a sorting key. For multi-step logic, a normal def function is usually clearer.

Practice

Use lambda to sort words by their length.

Study Loop

- Predict the output before running the example.
- Change one value and observe what happens.
- Explain the result in your own words.
- Write a small variation without copying the example.



25. map()

Explanation

map() applies a function to each item. In Python 3 it returns an iterator, so list() is commonly used when you want to materialise all results.

Example

```
nums=[1,2,3,4]
result=map(lambda n:n*10,nums)
print(list(result))
```

Output

```
[10, 20, 30, 40]
```

Map Thinking

Think: input items → transformation → output items. A list comprehension often expresses simple transformations even more directly.

Practice

Use map() to convert ['1','2','3'] into integers.

Study Loop

- Predict the output before running the example.
- Change one value and observe what happens.
- Explain the result in your own words.
- Write a small variation without copying the example.



26. filter()

Explanation

filter() keeps items for which a condition is true. Like map(), it returns an iterator in Python 3.

Example

```
nums=[1,2,3,4,5,6]
result=filter(lambda n:n%2==0,nums)
print(list(result))
```

Output

```
[2, 4, 6]
```

Filter Thinking

Think: input items $\rightarrow$ condition $\rightarrow$ selected items. For simple cases, a comprehension may be more readable.

Practice

Filter a list so only values greater than 50 remain.

Study Loop

- Predict the output before running the example.
- Change one value and observe what happens.
- Explain the result in your own words.
- Write a small variation without copying the example.



27. enumerate()

Explanation

enumerate() gives both an item and its position during iteration. It avoids manually maintaining a counter and keeps the counter synchronized with the sequence.

Example

```
langs=["Python","Java","Go"]
for i,lang in enumerate(langs,start=1):
    print(i,lang)
```

Output

```
1 Python
2 Java
3 Go
```

enumerate Tip

start=1 is useful for human-readable numbering even though Python indexes normally begin at 0.

Practice

Print a numbered list of five topics using enumerate().

Study Loop

- Predict the output before running the example.
- Change one value and observe what happens.
- Explain the result in your own words.
- Write a small variation without copying the example.



28. zip()

Explanation

zip() combines related values position by position. It is useful when two lists represent parallel data, such as names and scores.

Example

```
names=["Asha","Ravi","Om"]
scores=[88,76,91]
for name,score in zip(names,scores):
    print(name,score)
```

Output

```
Asha 88
Ravi 76
Om 91
```

zip Tip

zip() stops at the shortest input by default. Make sure the lists represent the relationship you intend.

Practice

Use zip() to create a dictionary from two lists of product names and prices.

Study Loop

- Predict the output before running the example.
- Change one value and observe what happens.
- Explain the result in your own words.
- Write a small variation without copying the example.



29. any() and all()

Explanation

any() answers whether at least one item is truthy; all() answers whether every item is truthy. They can make validation rules concise and expressive.

Example

```
marks=[72,81,64]
print(any(m>=90 for m in marks))
print(all(m>=40 for m in marks))
```

Output

```
False
True
```

Boolean Tools

These functions work well with generator expressions, which produce values as needed rather than creating an intermediate list.

Practice

Check whether all temperatures are below 50 and whether any temperature is below 0.

Study Loop

- Predict the output before running the example.
- Change one value and observe what happens.
- Explain the result in your own words.
- Write a small variation without copying the example.



30. Modules + Final Review

Explanation

A module is a Python file containing reusable code. Modules help separate responsibilities, reduce repetition and keep larger projects understandable. The main guard lets a file behave differently when executed directly versus imported.

Example

```
def main():  
    print("Program started")  
  
if __name__=="__main__":  
    main()
```

Output

Program started

Part 3 will move into Object-Oriented Python: classes, objects, constructors, inheritance and reusable application.

When the file is run directly, `__name__` is usually `'__main__'`. When imported, the main block does not automatically execute.

Practice

MINI CHALLENGE: build a student report using dictionaries, a function for averages, `enumerate()` for numbering, an f-string for output, and a separate helper module.

Study Loop

- Predict the output before running the example.
- Change one value and observe what happens.
- Explain the result in your own words.
- Write a small variation without copying the example.

