

Python Lists, Decisions, Loops and Functions

Hold many values in lists and dictionaries, make decisions with if, repeat work with loops, package steps into functions, and combine them into a small real program.

20 min read · Content last verified 20 October 2026 · Reviewed by Shravya M

Free awareness guide. Not legal or financial advice. <https://www.shravyadigital.com/guides/python-lists-loops-functions>

CONTENTS

1. Before you start
2. Chapter 1: Lists: many values in one place
3. Chapter 2: Dictionaries: values with labels
4. Chapter 3: Making decisions with if
5. Chapter 4: Repeating with loops
6. Chapter 5: Functions: reusable steps
7. Chapter 6: Putting it together
8. Chapter 7: Good habits
9. Key words

Before you start

This guide is the reading for **Course 2** of the Python for AI Pathway. It takes about 20 minutes and builds on Course 1, *Python First Steps*. Open a Google Colab notebook and try each example as you read.

You'll learn the four tools behind almost every program: **lists** to hold many values, **dictionaries** to label them, **if** to make decisions, **loops** to repeat, and **functions** to reuse your work.

All names and examples here are made up.

Chapter 1: Lists: many values in one place

A **list** holds several values, in order, inside square brackets:

```
marks = [72, 85, 64, 91, 58]
print(marks)
print(len(marks))
```

Output:

```
[72, 85, 64, 91, 58]
5
```

`len()` gives the number of items. Each item has a position called an **index**, and **counting starts at 0**:

```
marks = [72, 85, 64, 91, 58]
print(marks[0])
print(marks[-1])
```

Output:

```
72
58
```

`marks[0]` is the first item; `marks[-1]` is the last. You can add items with `append` :

```
cities = ["Delhi", "Chennai"]
cities.append("Kolkata")
print(cities)
```

Output:

```
['Delhi', 'Chennai', 'Kolkata']
```

Python also has quick helpers for lists of numbers: `sum()`, `max()`, `min()` :

```
marks = [72, 85, 64, 91, 58]  
print(sum(marks), max(marks), min(marks))
```

Output:

```
370 91 58
```

Chapter 2: Dictionaries: values with labels

A **dictionary** stores pairs of a **key** and a **value**, inside curly brackets. It's like a form with labelled boxes:

```
student = {"name": "Ravi", "class": 10, "city": "Indore"}  
print(student["name"])  
student["marks"] = 88  
print(student)
```

Output:

```
Ravi  
{'name': 'Ravi', 'class': 10, 'city': 'Indore', 'marks': 88}
```

You look a value up by its key, such as `student["name"]`, and add a new pair by assigning to a new key. Lists and dictionaries are how most data is held in Python, and in AI tools too.

Chapter 3: Making decisions with if

`if` runs code only when a condition is True. `elif` ("else if") checks another condition, and `else` covers everything left:

```
marks = 72
if marks >= 75:
    print("Distinction")
elif marks >= 40:
    print("Pass")
else:
    print("Needs another try")
```

Output:

```
Pass
```

Two rules matter:

- The line with `if`, `elif` or `else` ends with a **colon** `:`.
- The code that belongs to it is **indented** (moved right, usually 4 spaces). Python uses indentation to know what's inside the `if`.

You can combine conditions with `and`, `or` and `not`:

```
age = 19
has_id = True
print(age >= 18 and has_id)
```

Output:

```
True
```

Chapter 4: Repeating with loops

A **for loop** repeats code for every item in a list:

```
expenses = [1200, 450, 3000]
for amount in expenses:
    print("Spent", amount)
```

Output:

```
Spent 1200
Spent 450
Spent 3000
```

`range(n)` gives the numbers 0 up to $n-1$, which is handy for repeating a fixed number of times:

```
for i in range(3):
    print("Round", i)
```

Output:

```
Round 0
Round 1
Round 2
```

Loops and **if** together are powerful. Count how many students passed:

```
marks = [72, 85, 34, 91, 28]
passed = 0
for m in marks:
    if m >= 40:
        passed = passed + 1
print(passed, "students passed")
```

Output:

3 students passed

A **while loop** repeats as long as a condition stays True. Use it carefully: if the condition never becomes False, the loop runs for ever.

Chapter 5: Functions: reusable steps

A **function** is a named set of steps you can use again and again. You **define** it with `def`, and it can **return** a result:

```
def grade(marks):  
    if marks >= 75:  
        return "Distinction"  
    elif marks >= 40:  
        return "Pass"  
    return "Fail"  
  
print(grade(80))  
print(grade(52))  
print(grade(12))
```

Output:

```
Distinction  
Pass  
Fail
```

`marks` here is a **parameter**: the input the function works on. Write a function once, then call it with any value. You've already used built-in functions: `print`, `len`, `sum`.

Functions can take several parameters, and give them default values:

```
def total_with_gst(price, rate=18):  
    return price + price * rate / 100  
  
print(total_with_gst(1000))  
print(total_with_gst(1000, 5))
```

Output:

```
1180.0  
1050.0
```

Chapter 6: Putting it together

A small, realistic program: a monthly budget check.

```
budget = {"rent": 12000, "food": 6000, "travel": 2000}
spent = {"rent": 12000, "food": 7400, "travel": 1500}

def check(category):
    over = spent[category] - budget[category]
    if over > 0:
        return f"{category}: over by Rs {over}"
    return f"{category}: within budget"

for category in budget:
    print(check(category))
```

Output:

```
rent: within budget
food: over by Rs 1400
travel: within budget
```

It uses dictionaries for the data, a function for the rule, a loop to go through every category, and `if` to decide. That's the shape of most real programs, including the data work in Course 3.

Chapter 7: Good habits

- **Use clear names:** `monthly_rent` beats `x`.
- **Keep functions small:** one function, one job.
- **Comment the "why",** not the obvious.
- **Test with small examples first,** then with real data.
- **Watch the indentation:** mixing it up is the most common beginner error in Python.
- **Never paste passwords or keys into code** you share or upload. Course 5 shows the safe way.

Key words

- **List:** an ordered collection, `[1, 2, 3]` ; the first index is 0.
- **Index:** an item's position in a list.
- **Dictionary:** key–value pairs, `{"name": "Ravi"}` .
- **if / elif / else:** run code depending on a condition.
- **Indentation:** spaces at the start of a line that show which code belongs together.
- **for loop:** repeat code for each item.
- **range(n):** the numbers 0 to n–1.
- **while loop:** repeat while a condition is True.
- **Function / def / return:** a named, reusable set of steps that gives back a result.
- **Parameter:** an input a function takes.

Take the course for what you just read

Take the free 10-question assessment and pass with 8 or more. Every course you pass counts towards a Pathway Certificate issued by Shravya Digital, earned when you pass all five courses in the pathway: <https://digital.shravyaworld.org/courses/python-lists-loops-functions>

Sources

Checked against these primary sources on 20 October 2026. Rules change: if something here looks out of date, trust the source.

- Python documentation: the official Python tutorial (<https://docs.python.org/3/tutorial/>)
- Python documentation: data structures (<https://docs.python.org/3/tutorial/datastructures.html>)
- Python documentation: control flow and functions (<https://docs.python.org/3/tutorial/controlflow.html>)