VISTA HUB · CODING BASICS 2026

# Python Syntax Cheat Sheet for Beginners

Variables and types · collections · control flow
Functions · modules · built-ins · practical patterns
Print-friendly quick reference

FREE PRINTABLE PDF · A4 · ENGLISH

**Vista Hub · vhoriginal.com**

Original educational guide · Python 3 syntax

START HERE

# Python’s Small Building Blocks

Python code is read from top to bottom. Names point to values, indentation groups related statements, and a colon introduces a block such as an **if** statement, loop, function, or class. Save a file with the `.py` extension and run it with Python 3.

> **Core rule:** Python uses indentation as syntax. Use four spaces for each level, keep the indentation consistent, and do not mix tabs and spaces in the same block.

## Variables and common types

| Type | Example | What it represents | Useful check |
|---|---|---|---|
| **int** | `count = 12` | Whole numbers | `type(count)` |
| **float** | `price = 4.50` | Decimal numbers | `isinstance(price, float)` |
| **str** | `name = "Mina"` | Text in quotes | `len(name)` |
| **bool** | `ready = True` | True or False | `bool(value)` |
| **None** | `result = None` | No value yet | `result is None` |

```
# Assignment creates or updates a name
student = "Ari"
age = 19
is_beginner = True

# Convert between simple types
year_text = "2026"
year = int(year_text)
label = str(year)
print(student, year, is_beginner)
```

## Input and output

### print()

Displays values. Separate arguments with commas, or join them in an f-string. Use sep= and end= when you need control.

### input()

Reads text from the user. Convert it when a number is expected: `age = int(input("Age: "))`.

> **Naming habit:** Use lowercase `snake_case` for variables and functions, descriptive names for values, and uppercase names for constants such as `MAX_RETRIES`.

WORK WITH DATA

# Strings and Collections

## Strings: text you can inspect and shape

| Expression | Result or purpose |
|---|---|
| `text[0]` | First character; indexes start at zero. |
| `text[-1]` | Last character. |
| `text[1:4]` | Slice from index 1 up to, not including, 4. |
| `text.lower() / text.upper()` | Return lowercase or uppercase text. |
| `text.strip()` | Remove surrounding whitespace. |
| `text.split(",")` | Split text into a list at each comma. |
| `", ".join(words)` | Join an iterable of strings. |

```
title = "  Python basics  "
clean = title.strip()
words = clean.split()
print(clean[0], clean[-1], " / ".join(words))
```

## Four built-in collection shapes

| Collection | Literal | Order / change | Typical use |
|---|---|---|---|
| **list** | `["a", "b"]` | Ordered, mutable | Items that may grow or change |
| **tuple** | `("x", "y")` | Ordered, immutable | Fixed grouped values |
| **dict** | `{"id": 7}` | Key → value mapping | Named fields or lookup tables |
| **set** | `{"red", "blue"}` | Unique items; no index | Membership and de-duplication |

### Lists

```
items.append("new")
items.pop()
items[0] = "changed"
len(items)
```

### Dictionaries

```
user["name"]
user.get("email")
user["active"] = True
user.keys()
```

### Tuples

```
point = (3, 5)
x, y = point
```

### Sets

```
tags.add("python")
"python" in tags
a | b # union
a & b # intersection
```

> **Mutability shortcut:** lists, dictionaries, and sets can change in place. Strings and tuples cannot; operations create a new value instead.

MAKE DECISIONS

# Operators and Conditional Logic

## Operator quick table

| Group | Operators | Example | Meaning |
|---|---|---|---|
| Arithmetic | + − * / // % ** | 7 // 2 → 3 | Add, subtract, multiply, divide, floor divide, remainder, power |
| Comparison | == != < <= > >= | score >= 50 | Compare values; result is Boolean |
| Logic | and, or, not | age >= 18 and active | Combine or invert conditions |
| Membership | in, not in | "py" in word | Check an item or substring |
| Identity | is, is not | value is None | Check whether objects are the same object |
| Assignment | = += -= *= | total += 5 | Store or update a value |

**Equality versus identity:** Use `==` to compare values. Use `is None` for the special singleton `None`; do not use `is` as a general value comparison.

## if / elif / else

```
temperature = 21
if temperature >= 30:
    message = "hot"
elif temperature >= 18:
    message = "comfortable"
else:
    message = "cool"
print(message)
```

## Truthiness and a compact conditional

### False-like values

`False`, `None`, numeric zero, and empty strings, lists, tuples, sets, and dictionaries act as false in a condition. Most other values act as true.

### Conditional expression

```
label = "adult" if age >= 18 else "minor"
```

Use it only when the expression stays easy to read.

**Readability check:** Put the normal path in a clear branch, keep conditions short, and use parentheses when a combined expression needs visual grouping.

REPEAT WITH PURPOSE

# for and while Loops

## for: visit each item

Use a `for` loop when you have an iterable such as a list, string, dictionary, set, or `range()`. Python assigns the next item to the loop variable on each pass.

```
scores = [72, 88, 91]
for score in scores:
    if score >= 90:
        print(score, "excellent")
    else:
        print(score, "keep practicing")
```

| Pattern | What it does |
|---|---|
| `range(5)` | 0, 1, 2, 3, 4 |
| `range(2, 7)` | 2 through 6 |
| `range(0, 10, 2)` | 0, 2, 4, 6, 8 |
| `enumerate(items, start=1)` | Pairs each item with a counting number. |
| `for key, value in data.items()` | Iterates through dictionary pairs. |

```
for position, item in enumerate(["tea", "code"], start=1):
    print(position, item)

for number in range(1, 6):
    if number == 3:
        continue  # skip this pass
    if number == 5:
        break     # leave the loop
```

## while: repeat while a condition is true

```
attempts = 0
while attempts < 3:
    attempts += 1
    print("Attempt", attempts)
```

> **Prevent infinite loops:** A `while` loop needs a condition that can eventually become false, or a clear `break`. Prefer `for` when the number of items or passes is already known.

REUSE YOUR THINKING

# Functions, Return Values, and Imports

## Functions

```
def greet(name, punctuation="!"):
    """Return a short greeting."""
    return f"Hello, {name}{punctuation}"

message = greet("Ari")
print(message)
```

| Function part | Syntax / idea |
|---|---|
| Define | `def name(parameters):` |
| Parameter | Name in the definition; receives an argument. |
| Argument | Actual value supplied in the call. |
| Default | `punctuation="!"` |
| Return | Send a value back; without it, the function returns `None`. |
| Scope | Names created inside a function are local by default. |

> **Good function test:** Give one function one clear job. Name it with a verb, keep inputs and outputs obvious, and return data instead of printing when another part of a program may need the result.

## Modules and imports

### Import a module

```
import math
math.sqrt(25)
```

### Use an alias

```
import datetime as dt
dt.date.today()
```

### Import a name

```
from pathlib import Path
Path("notes.txt")
```

### Make a module

Put reusable code in another `.py` file, then import it from a script in the same project.

```
if __name__ == "__main__":
    print("Run this file directly")
```

**Why this guard?** The block runs when the file is launched directly, but not when the file is imported as a module.

## Simple error handling

```
try:
    number = int(input("Number: "))
except ValueError:
    print("Please enter a whole number.")
```

EVERYDAY PATTERNS

# Built-ins, f-Strings, Comprehensions, and Files

## Common built-ins worth remembering

| Built-in | Quick use | Purpose |
|---|---|---|
| `len(x)` | `len(items)` | Number of items or characters |
| `type(x)` | `type(value)` | Inspect the type |
| `int / float / str` | `float("3.2")` | Convert a value |
| `range()` | `range(3)` | Create a number sequence |
| `min / max / sum` | `sum(scores)` | Simple numeric summaries |
| `sorted(x)` | `sorted(names)` | Return a sorted list |
| `zip(a, b)` | `zip(names, scores)` | Pair items by position |
| `any / all` | `all(checks)` | Test a group of Boolean values |

## f-strings: readable formatted text

```
name = "Mina"
score = 87
print(f"{name} scored {score}%")
print(f"Average: {score / 2:.1f}")
```

## Short list comprehensions

A comprehension builds a collection in one readable expression. Use a normal loop when the logic becomes crowded.

```
numbers = [1, 2, 3, 4, 5]
squares = [n * n for n in numbers]
evens = [n for n in numbers if n % 2 == 0]
```

## Open a text file safely

```
with open("notes.txt", "r", encoding="utf-8") as file:
    text = file.read()

with open("log.txt", "w", encoding="utf-8") as file:
    file.write("First line\n")
```

**Modes**

`"r"` reads, `"w"` writes and replaces, `"a"` appends. Use `"x"` to create only if the path does not exist.

**Why with?**

The context manager closes the file even if an error occurs. Always provide an encoding for predictable text handling.

KEEP BESIDE YOUR EDITOR

# One-Page Syntax Recap

### Values

```
x = 10
name = "Ari"
ready = True
missing = None
```

### Collections

```
items = []
record = {}
coords = (3, 5)
unique = set()
```

### Decision

```
if condition:
do_this()
elif other:
do_that()
else:
finish()
```

### Loop

```
for item in items:
print(item)

while ready:
work()
```

### Function

```
def add(a, b=0):
return a + b
```

### Import

```
import math
from pathlib import Path
Path("file.txt")
```

## Mini practice prompt

**Build a tiny reading tracker:** Store book titles in a list, ask for one new title, append it, print the total with `len()`, and display each title with `enumerate()`. Add a function that prints a friendly summary.

| Before you run | Quick check |
|---|---|
| Indentation | Every block after a colon is indented consistently. |
| Names | Variables and functions use clear `snake_case` names. |
| Types | Input is text until you explicitly convert it. |
| Loops | A `while` loop can make progress toward stopping. |
| Files | Text files use `with open(..., encoding="utf-8")`. |
| Errors | Read the traceback from the bottom up; fix the first useful line. |

**Next step:** Change one example, run it, observe the output, and make a small prediction before trying again. Short experiments build syntax memory faster than copying a long script without testing it.

**Read it. Type it. Run it.**

Vista Hub · Original Python Syntax Cheat Sheet for Beginners · 2026