Built-in Functions

Introduction

Python provides a rich set of **built-in functions** that are always available without importing any module.

These functions cover common tasks — **type conversion**, **numeric operations**, **sequence handling**, **I/O**, **introspection**, and more.

Table of contents

01 Basic I/O & Interaction

02 Sequence & Collection Helpers

03 Functional-Style Tools

04 Numeric & Math Utilities

05 Type Conversion & Checking

06 File & Dynamic Execution

01

Basic I/O & Interaction

Basic I/O & Interaction

01 — `print()` — Write one or more objects to the console

02 — `input()` — Read a line of text from the user as a string

Basic I/O – print()

- **Prints** one or more objects to the console
- **Syntax**: print(*objects, sep=' ', end='\n')
- **Returns**: None

Python	**Output**
`print("Hello", "World", sep="-")`	`Hello-World`

Basic I/O – input()

- **Reads** a line of text from user input.
- **Syntax**: input(prompt='')
- **Returns**: String

Python

```
name = input("Enter your name: ")
print(name)
```

02

Sequence & Collection Helpers

Sequence & Collection Helpers

01	`len()`	Returns the number of items in a container
02	`range()`	Generates a sequence of numbers.
03	`list()`, `tuple()`, `set()`, `dict()`	Create list, tuple, set, or dictionary from an iterable.

Sequence & Collection Helpers

04 — `enumerate()` — yield pairs of (index, value) when iterating

05 — `zip()` — aggregate elements from multiple iterables into tuples

06 — `sorted()` — return a new sorted list from any iterable

07 — `reversed()` — return an iterator over the elements of a sequence in reverse order

Sequence & Collection Helpers – len()

- Returns the **number of items** in a container
- **Syntax**: len(obj)
- **Returns**: Integer

Python	**Output**
`len([1, 2, 3, 4])`	`4`

Sequence & Collection Helpers – range()

- **Generates a sequence** of numbers
- **Syntax**: range(start, stop[, step])
- **Returns**: Range object (iterable)

Python	**Output**
`for i in range(1, 5, 2):` `  print(i)`	`1 3`

Sequence & Collection Helpers – list(), tuple(), set()

- Create **list, tuple, or set**, from an iterable
- **Syntax**: list(iterable), tuple(iterable), set(iterable)
- **Returns**: Respective collection type

Python	Output

`print(list("abc"))`
`print(tuple("abc"))`
`print(set("abc"))`
	`['a', 'b', 'c']`
	`('a', 'b', 'c')`
	`{'b', 'c', 'a'}`

Sequence & Collection Helpers – dict()

- Create **dictionary** from an iterable
- **Syntax**: dict(**kwargs)
- **Returns**: Respective collection type

Python	Output
`dict(a=1, b=2)`	`{'a': 1, 'b': 2}`

Sequence & Collection Helpers – enumerate()

- Returns an **iterator that produces tuples** of (**index, item**).
- **Syntax**: enumerate(iterable, start=0)
- **Returns**: Enumerate object

Python	**Output**

`for index, value in enumerate(['a', 'b', 'c']):`
`  print(index, value)`
	`0 a`
	`1 b`
	`2 c`

Sequence & Collection Helpers – zip()

- **Aggregates** elements from **multiple iterables** into tuples.
- **Syntax**: zip(iterable1, iterable2, ...)
- **Returns**: Iterator of tuples

Python	**Output**
`list(zip([1, 2], ['a', 'b']))`	`[(1, 'a'), (2, 'b')]`

Sequence & Collection Helpers – sorted()

- Returns a **sorted list** from the iterable
- **Syntax**: sorted(iterable, key=None, reverse=False)
- **Returns**: List

Python	Output
`sorted([3, 1, 2])`	`[1, 2, 3]`

Sequence & Collection Helpers – reversed()

- Returns a **reverse** iterator
- **Syntax**: reversed(seq)
- **Returns**: Reverse iterator

Python	Output
`list(reversed([1, 2, 3]))`	`[3, 2, 1]`

03

Functional-Style Tools

Functional-Style Tools

01 — `map()` — apply a function to each item of an iterable and return an iterator of results

02 — `filter()` — select items from an iterable for which a predicate function returns True

03 — `any()` — return True if at least one element of an iterable is truthy

04 — `all()` — return True only if every element of an iterable is truthy

Functional-Style Tools – map()

- **Applies a function** to all items in an iterable.
- **Syntax**: map(function, iterable)
- **Returns**: Iterator

Python	Output
`list(map(str.upper, ['a', 'b', 'c']))`	`['A', 'B', 'C']`
`list(map(lambda x: x ** 2, [1,2,3,4]))`	`[1, 4, 9, 16]`

Functional-Style Tools – filter()

- **Filters items** where function returns **True**.
- **Syntax**: filter(function, iterable)
- **Returns**: Iterator

Python	**Output**
`list(filter(lambda x: x % 2 == 0, [1, 2, 3, 4]))`	`[2, 4]`

Functional-Style Tools – any()

- **Returns True** if at **least one element** is **truthy**.
- **Syntax**: any(iterable)
- **Returns**: Boolean

Python	Output
`any([0, False, 5])`	`True`
`any(x > 0 for x in [-1, -2, 3])`	`True`

Functional-Style Tools – all()

- **Returns True** if **all element** is **truthy**.
- **Syntax**: all(iterable)
- **Returns**: Boolean

Python	Output
`all([0, False, 5])`	`False`
`all(x > 0 for x in [1, 2, 3])`	`True`

04

Numeric & Math Utilities

Numeric & Math Utilities

01 — `sum()` — add up all items in an iterable (optionally starting from a given value)

02 — `min()` `max()` — find the smallest or largest item in an iterable (or among multiple arguments)

03 — `abs()` — compute the absolute value of a number

Numeric & Math Utilities

04 — `round()` — round a number to a given number of decimal places

05 — `pow()` — raise a number to a power (with optional modulus argument)

06 — `divmod()` — return a tuple of (quotient, remainder) for integer division

Numeric & Math Utilities – sum()

- Returns the **sum of all elements** in an iterable.
- **Syntax**: sum(iterable, start=0)
- **Returns**: Number

Python	Output
`sum([1, 2, 3, 4])`	`10`

Numeric & Math Utilities – min(), max()

- Returns the **smallest** or **largest** element.
- **Syntax**: min(iterable, *[, key]), max(iterable, *[, key]))
- **Returns**: Number

Python	Output
`min([3, 1, 4])`	`1`
`max([3, 1, 4])`	`4`
`print(min(["hi", "hello", "python"]))`	`Hello`
`print(max(["hi", "hello", "python"]))`	`python`

Numeric & Math Utilities – abs()

- Returns **absolute value**.
- **Syntax**: abs(x)
- **Returns**: Absolute value

Python	Output
`abs(-7)`	7

Numeric & Math Utilities – round()

- Rounds number to **nearest integer** or **specified decimal places**
- **Syntax**: round(x[, ndigits])
- **Returns**: Rounded number

Python	Output
`round(3.14159, 2)`	`3.14`

Numeric & Math Utilities – pow()

- Raises **number** to a **power**.
- **Syntax**: pow(x, y[, mod])
- **Returns**: Number

Python	**Output**
`pow(2, 3)`	8

Numeric & Math Utilities – divmod()

- Returns **quotient** and **remainder**.
- **Syntax**: divmod(a, b)
- **Returns**: Tuple (quotient, remainder)

Python	Output
`divmod(17, 5)`	`(3, 2)`

05

Type Conversion & Checking

Type Conversion & Checking

01 — `int()`, `float()`, `str()`, `bool()` — convert values to integer, floating-point, string, or boolean types

02 — `type()`, `isinstance()`, `issubclass()` — inspect and test types or class relationships

Type Conversion & Checking - int(), float(), str(), bool()

- Convert to **integer, float, string, or Boolean**
- **Syntax**: int(x), float(x), str(x), bool(x)
- **Returns**: Converted value

Python	Output
`int("5")`	`5`
`bool(0)`	`False`

Type Conversion & Checking - type(), isinstance(), issubclass()

- **Type checking** and **inspection**
- **Syntax**: type(obj), isinstance(obj, cls), issubclass(cls, clsinfo)
- **Returns**: Boolean or Type

Python	**Output**
`isinstance(5, int)`	`True`
`type("hello")`	`<class 'str'>`

06

File & Dynamic Execution

File & Dynamic Execution

01 — `open()` — open a file and return a file object for reading or writing

02 — `eval()` — evaluate a Python expression given as a string and return its value

03 — `exec()` — execute dynamically generated Python code

File & Dynamic Execution – open()

- **Opens** a file and **returns** a file object.
- **Syntax**: open(file, mode='r')
- **Returns**: File object

```python
f = open('file.txt', 'r')
```

File & Dynamic Execution – eval()

- **Evaluates** a Python **expression** from a **string**.
- **Syntax**: eval(expression)
- **Returns**: Result of expression

Python	Output
`eval("2 + 3")`	`5`

File & Dynamic Execution – exec()

- **Executes** dynamically created **Python code**.
- **Syntax**: exec(object)
- **Returns**: None

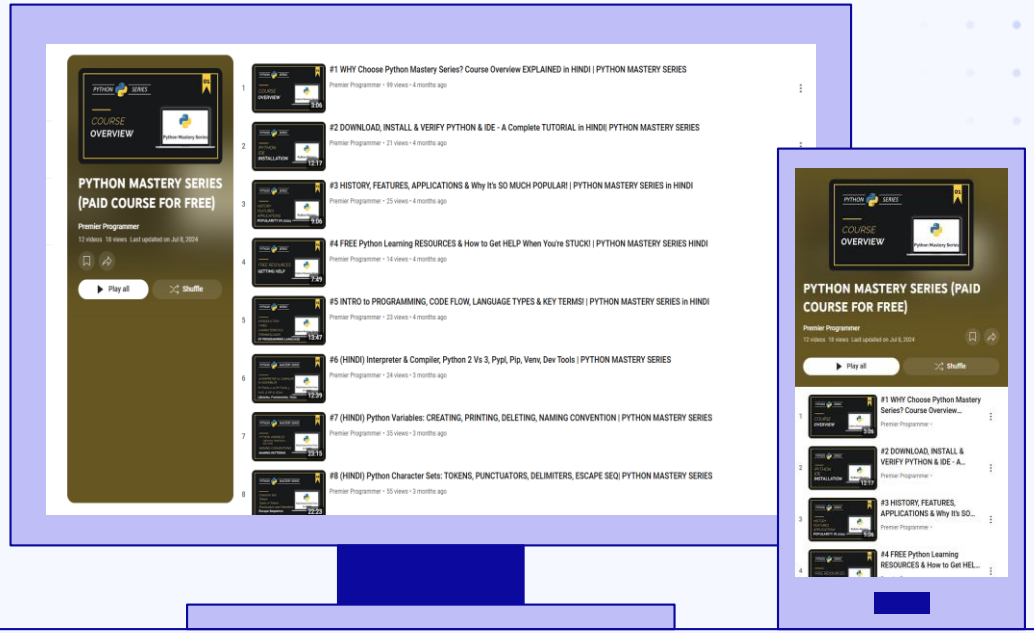
Python	**Output**
`exec("a = 5; print(a)")`	5

Practice Set - 4

Download Link in **Description** and **Pinned Comment**

WATCH

Level up your coding with each episode in this focused Python series.



Next Video!

Practice Set – 4
Solution