📘 Chapter 16: Built-in Functions – From Basics to Power Tools

🎯 What You Will Learn

- Understand what built-in functions are and why Python includes them
- Use core built-in functions like `len()`, `sum()`, `min()` in programs
- Apply utility functions like `zip()`, `enumerate()`, `all()` for cleaner code
- Master advanced tools like `map()`, `filter()`, `lambda`, and `sorted()`
- Know when to avoid risky functions like `eval()` and `exec()`
- Use `dir()`, `help()` to explore Python more confidently

🧠 Why Built-in Functions Matter

Python's built-in functions are **pre-defined helpers** that perform common tasks efficiently.
Instead of writing code from scratch, you can call these tools directly and write **cleaner, shorter, and faster** logic.
From data processing to functional programming, built-ins form the **foundation of Python fluency**.

🔷 Concept Introduction

What Are Built-in Functions?

They are functions that are **always available** in Python, no import needed.

Examples:

```
print("Hello")       # prints to screen
len("hello")         # returns 5
sum([1, 2, 3])       # returns 6
```

These are optimized, battle-tested functions that Python provides by default.

🔁 20 Must-Know Built-in Functions

🟢 Basic Utilities

1. `print()` – Output to console

```
print("Hello, world!")
```

2. `len()` – Length of string, list, etc.

```
len("Python")  # 6
```

3. `type()` – Find data type

```
type(3.5)  # <class 'float'>
```

4. `input()` – Take user input

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

5. `abs()` – Absolute value

```
abs(-7)  # 7
```

6. `round()` – Round numbers

```
round(4.6)  # 5
```

7. `sum()` – Add items in iterable

```
sum([10, 20, 30])  # 60
```

8. `min()` / `max()` – Smallest / Largest

```
min([2, 3, 1])  # 1
max("zebra")    # 'z'
```

9. `sorted()` – Returns a new sorted list

```
sorted([3, 1, 2])  # [1, 2, 3]
```

10. `reversed()` – Reverse iterable

```
list(reversed([1, 2, 3]))  # [3, 2, 1]
```

🟡 Utility & Logic Helpers

11. `all()` – True if all items are truthy

```
all([True, 1, "yes"])  # True
```

12. `any()` – True if *any* item is truthy

```
any([False, 0, "", 5])  # True
```

13. `enumerate()` – Index + item

```
for i, val in enumerate(["a", "b"]):
    print(i, val)
```

14. `zip()` – Pair elements

```
list(zip([1,2], ['a','b']))  # [(1,'a'), (2,'b')]
```

15. `range()` – Sequence generator

```
for i in range(3): print(i)  # 0 1 2
```

🔴 Advanced Functional Programming

16. `map()` – Apply function to all items

```
list(map(str.upper, ["a", "b"]))  # ['A', 'B']
```

17. `filter()` – Keep only True results

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

18. `lambda` – Inline anonymous function

```
add = lambda x, y: x + y
add(2, 3)  # 5
```

19. `eval()` – Evaluate string as code (⚠ Dangerous!)

```
eval("2 + 3")  # 5
```

20. `exec()` – Execute full Python code (⚠️ Risky!)

```
exec("x = 5")
print(x)  # 5
```

🧪 Mini Quiz (15 Questions)

1. What does `len([1, 2, 3])` return?
2. What is the difference between `print()` and `return`?
3. Predict the output: `type("5")`
4. Which function checks if all values are truthy?
5. What is the result of `sum([])`?
6. What is the use of `enumerate()`?
7. What's the output of `sorted("cab")`?
8. Use `zip()` to combine `[1,2]` and `['a','b']`
9. What does `lambda x: x*2` return when called with `3`?
10. Is `eval("2+2")` safe?
11. What's the result of `all([1, 0, True])`?
12. Use `filter()` to keep even numbers.
13. Predict output: `list(map(str.upper, ["a", "b"]))`
14. How does `input()` return values?
15. What will `reversed("abc")` return?

🧩 Basic Practice Problems (15)

1. Print your name using `print()`.
2. Ask user age and display it.
3. Use `len()` on your full name.
4. Use `sum()` to total 3 numbers.
5. Sort a list of 5 random numbers.
6. Round a float to nearest integer.
7. Use `abs()` on -15.
8. Use `type()` on `True`.
9. Use `any()` on `[False, 0, 3]`.
10. Find max of 5 values.
11. Pair two lists using `zip()`.
12. Reverse a string using `reversed()`.

13. Use `map()` to square numbers.
14. Filter out odd numbers.
15. Use `enumerate()` to show index and value.

🚀 Intermediate Practice (7)

1. Use `map()` + `lambda` to double all values.
2. Use `filter()` to remove empty strings.
3. Use `zip()` to match countries and capitals.
4. Create a calculator with `eval()` (warn about risks).
5. Ask a user input, and reverse it.
6. Print only even-indexed letters of a string using `enumerate()`.
7. Build a one-liner function using `lambda` that checks if a number is even.

🔧 Debug Challenges (7)

1. What's wrong here?

```
print(len)
```

2. Predict error:

```
sum(1, 2, 3)
```

3. Fix it:

```
list(map(lambda x: x * 2, 5))
```

4. What will this return?

```
list(filter(None, [0, 1, "", "a"]))
```

5. Explain the issue:

```
eval("import os")
```

6. Rewrite using `lambda`:

```
def add(x): return x + 2
```

7. Logic bug:

```
max(4, 2, "7")
```

✅ Summary: When to Use Built-in Functions

- Use built-ins to write concise, efficient Python code.
- `sum`, `min`, `max`, `sorted` are better than manual loops.
- Use `map`, `filter`, `lambda` for functional transformations.
- Avoid `eval()` / `exec()` unless absolutely necessary.
- Explore unknown objects with `dir()` and `help()`.

📋 TL;DR Technique Table

Function	Purpose	Example
`len()`	Get length	`len("abc") → 3`
`sum()`	Add values	`sum([1,2,3]) → 6`
`map()`	Transform items	`map(str, [1,2])`
`filter()`	Keep matching items	`filter(odd, lst)`
`lambda`	Inline short function	`lambda x: x+1`
`enumerate()`	Index + value loop	`for i,v in enumerate(...)`
`zip()`	Combine iterables	`zip(keys, values)`

🔥 Bonus

- Use `dir()` to inspect any object:

```
dir("hello")
```

- Use `help()` for documentation:

```
help(str.upper)
```

- Avoid `eval()` if user input is involved.

End of Chapter 16 – Built-in Functions ✅