Python Cheatsheet for Beginners

By Ahnaf Chowdhury

Python Cheatsheet for Beginners

With Simple Explanations for Every Function

🖨️ 1. print() – Show Output

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

What it does: It shows text or values on the screen.

Use it to: See results, debug, or communicate with users.

💬 2. Comments – Notes in Your Code

Single-line comment

```
# This is a comment
```

Multi-line comment

```
"""
This is a
multi-line comment
"""
```

Why use them? To explain your code so you or others can understand it later.

Python ignores comments when running your program.

📦 3. Variables – Store Information

```
name = "Impala"       # String
age = 25              # Integer
height = 5.8          # Float
is_happy = True       # Boolean
```

What is a variable? A container for storing values.

You can store text, numbers, or True/False values.

🔢 4. Data Types – Kinds of Information

Data Type	Example	What It is
Int	10, -5	Whole number
float	3.14, 7.0	Decimal number
str	"hello"	Text
bool	True, False	Yes/No or On/Off logic

Use *type()* to check what type it is:

```
print(type(name))  # <class 'str'>
```

➕ 5. Math Operators – Do Calculations

```
x = 10
y = 3

x + y  # 13 (addition)
```

```
x - y  # 7  (subtraction)
x * y  # 30 (multiplication)
x / y  # 3.33 (division)
x // y # 3 (floor division)
x % y  # 1 (remainder)
x ** y # 1000 (power)
```

These are used to do math in Python.

🔤 6. Strings – Text Handling

```
greeting = "Hello"
print(greeting.upper())  # HELLO
print(greeting.lower())  # hello
print(len(greeting))  # 5
print(greeting[0])    # H
```

What is a string? A bunch of characters (text).

Indexing lets you pick one letter at a time.

Use f-strings to add variables into strings:

```
name = "Impala"
print(f"Hi, I'm {name}!")
```

📋 7. Lists – Store Many Values

```
fruits = ["apple", "banana", "mango"]
print(fruits[1])      # banana
fruits.append("grape")   # Add item
fruits.remove("banana")  # Remove item
```

A list holds many items.

It can grow or shrink as you add or remove things.

🔐 8. Tuples – Fixed Collections

```
person = ("John", 30)
print(person[0])  # John
```

Like a list, but you cannot change the items once created.

🗃️ 9. Dictionaries – Key-Value Pairs

```
person = {"name": "Alice", "age": 25}
print(person["name"])  # Alice
person["age"] = 26
```

A dictionary stores information with a key and a value.

Keys are like labels, values are the data.

🔁 10. Conditional Statements – Make Decisions

```
age = 20

if age >= 18:
      print("Adult")
else:
      print("Minor")
```

Use if, elif, else to choose what to do based on a condition.

🔁 11. Loops – Repeat Code

For loop

```
for i in range(5):
      print(i)  # 0 to 4
```

While loop

```
count = 0
while count < 3:
      print("Counting...")
      count += 1
```

When to use them? When you want to do something multiple times.

🛠️ 12. Functions – Reusable Code

```
def greet(name):
      print(f"Hello, {name}!")

greet("Impala")  # Hello, Impala!
```

A function is a block of code you can use over and over.

Define it with **def**, then call it when you need it.

🎤 13. input() – Get User Input

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

```
print("Welcome,", name)
```

This pauses the program and waits for user input.

It always returns a string, even if you type a number.

🔄 14. Type Conversion

```
age = input("Your age? ")  # string
age = int(age)             # convert to int
print(age + 1)
```

Use *int(), float(), str()* to change the type of a value.

🧰 15. Built-in Functions (Most Useful)

Function	What It Does
print()	Shows text or values
input()	Gets input from the user
len()	Counts how many items or characters
type()	Shows the data type of a value
range()	Makes a range of numbers for loops
int(), str(), float()	Convert between types

✅ 14. Built-in Functions

These are functions that are always available in Python, without needing to import anything.

print()

What it does: Displays text or values on the screen.

Example:

```
print("Hello")
```

shows Hello.

input()

What it does: Lets the user type something from the keyboard.

Returns: Always returns a string (even if you type a number).

Example:

```
name = input("What's your name? ")
print("Hello,", name)
```

len()

What it does: Tells you how many items are in a list, string, or dictionary.

Example:

```
len("hello")
```

returns 5.

type()

What it does: Tells you what kind of data a value is.

Example:

```
type(42)
```

returns <class 'int'>.

`str(), int(), float()`

What they do: Convert one type of value into another.

str(42) → '42'

int("5") → 5

float("3.14") → 3.14

`range()`

What it does: Generates a list of numbers (used often in loops).

Example:

```
range(5)
```

gives 0, 1, 2, 3, 4

✅ 15. File Handling

Python can read from and write to text files.

`open()`

What it does: Opens a file to read or write.

Modes:

"r" = read

"w" = write (erases file first)

"a" = append (adds to the end)

`read()`

Reads the entire contents of the file.

`write()`

Writes the given string into a file.

`with open(...) as …`

Best practice for working with files.

It automatically closes the file after the block.

✅ 16. Error Handling (Try-Except)

Used to handle situations that may cause your program to crash.

`try: ... except: …`

Tries to run the code inside try.

If an error occurs, it runs the code inside except.

Example:

```
try:
      result = 5 / 0
```

```
except ZeroDivisionError:
      print("You can't divide by zero.")
```

✅ 17. Importing Modules

Modules are collections of functions and tools.

`import math`

Loads the math module which has extra math functions like sqrt() and constants like pi.

`from math import pi, sqrt`

Only imports pi and sqrt() instead of the whole module.

✅ 18. Object-Oriented Programming (Classes and Objects)

Used to organize related code into objects with properties and actions.

`class`

A blueprint for creating objects.

`__init__()`

A special method that runs when you create a new object.

It initializes (sets up) the object's properties.

`self`

Refers to the object itself inside a class.

Example:

```
class Dog:
      def __init__(self, name):
      self.name = name
```

✅ 19. List Comprehensions

A quick way to create new lists.

Normal way:

```
squares = []
for x in range(5):
      squares.append(x**2)
```

List comprehension way:

```
squares = [x**2 for x in range(5)]
```

With condition:

```
evens = [x for x in range(10) if x % 2 == 0]
```

✅ 20. Lambda Functions

A lambda is a small one-line function with no name.

Example:

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

Used when you need a quick function just once, especially in functions like map() or filter().

✅ 21. map(), filter(), and reduce()

These functions apply logic to a list of items.

`map(function, list)`

Applies a function to each item in a list.

```
nums = [1, 2, 3]
squares = list(map(lambda x: x**2, nums))
```

`filter(function, list)`

Filters out items that don't meet a condition.

```
even = list(filter(lambda x: x % 2 == 0, nums))
```

`reduce(function, list)`

Repeatedly combines two items until one result is left.

```
from functools import reduce
total = reduce(lambda x, y: x + y, nums)
```

✅ 22. More Useful Functions

`sorted(list)`

Returns a new sorted list.

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

zip(list1, list2)

Combines elements from two or more lists.

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

enumerate(list)

Adds index numbers when looping.

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

any() and all()

any(): Returns True if any item is True.

all(): Returns True if all items are True.

```
any([False, True])  # True
all([True, True])   # True
```

✅ 23. Working with JSON

json.dumps()

Converts a Python object (like a dictionary) into a JSON string.

json.loads()

Converts a JSON string back into a Python object.

Example:

```
import json
data = {"name": "Impala", "age": 25}
json_str = json.dumps(data)
parsed = json.loads(json_str)
print(parsed["name"])  # Impala
```