Python CheatSheet (2025)

Last Updated : 03 Mar, 2025

Python is one of the most widely-used and popular programming languages, was developed by Guido van Rossum and released first in 1991. Python is a free and open-source language with a very simple and clean syntax which makes it easy for developers to **learn Python**. It supports object-oriented programming and is most commonly used to perform general-purpose programming.

Why Python?

- **Easy to Learn** – Clean, readable syntax that feels like plain English.
- **Free & Open-Source** – No cost, no restrictions—just pure coding freedom.
- **Object-Oriented & Versatile** – Supports multiple programming paradigms.
- **Massive Community Support** – Tons of libraries, frameworks and active contributors.

Where is Python Used?

- Data Science & Machine Learning – AI, analytics and automation.
- Web Development – Frameworks like Django & Flask for building web apps.
- Automation & Scripting – Simplifying repetitive tasks.
- Game Development – Used in game engines and AI-driven gaming.

Table of Content

To master python from scratch, you can refer to our article: Python Tutorial

Python Basics

Printing Output

print() function in Python is used to print Python objects as strings as standard output. keyword end can be used to avoid the new line after the output or end the output with a different string.

```
# python program to print "Hello World"
print("Hello World")
```

```
print("GeeksforGeeks", end=' ')
```

Python sep parameter in print()

The separator between the inputs to the print() method in Python is by default a space, however, this can be changed to any character, integer, or string of our choice. The 'sep' argument is used to do the same thing.

```
# code for disabling the softspace feature
print('09', '12', '2016', sep='-')

# another example
print('Example', 'geeksforgeeks', sep='@')
```

Python Input

input() method in Python is used to accept user input. By default, it returns the user input as a string. By default, the input() function accepts user input as a string.

```
# Python program showing
# a use of input()

val = input("Enter your value: ")
print(val)
```

Output:

```
Enter your value: Hello Geeks
Hello Geeks
```

Python Comment

Comments in Python are the lines in the code that are ignored by the interpreter during the execution of the program. There are three types of

- Single line Comments
- Multiline Comments
- Docstring Comments

```
# Single Line comment

# Python program to demonstrate
# multiline comments

""" Python program to demonstrate
 multiline comments"""

name = "geeksforgeeks"
print(name)
```

Output

```
geeksforgeeks
```

Variables and Data Types

Variables store values and Python supports multiple data types.

```
# DataType Output: int
x = 50

# DataType Output: float
x = 60.5

# DataType Output: complex
x = 3j
```

Type Checking and Conversion

We can check the data type of a variable using type() and convert types if needed.

```
print(type(x))  # <class 'int'>
y = int(y)      # Convert float to int
```

Operators in Python

In general, Operators are used to execute operations on values and variables. These are standard symbols used in logical and mathematical processes.

- Arithmetic Operators are used to perform mathematical operations .
- Comparison Operators compare values and return True or False based on the criteria.
- Logical operators are used on conditional statements in Python (either True or False).
- Bitwise operators are used in Python to do bitwise operations on integers.

Control Flow

Conditional Statements

```
if x > 5:
    print("x is greater than 5")
elif x == 5:
    print("x is 5")
else:
    print("x is less than 5")
```

Loops

Loops help iterate over sequences or repeat actions.

For Loop:

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

While Loop:

```
i = 0
while i < 5:
    print(i)
    i += 1
```

Loop Control Statements

Loop Control Statements include break and continue.

break exits the loop, while continue skips the current iteration.

```
for i in range(10):
    if i == 5:
        break  # Exits loop
    if i == 3:
        continue  # Skips iteration
    print(i)
```

Interesting Facts about Loops in Python:

```
vowels=['a','e','i','o','u']
for i, letter in enumerate(vowels):
    print (i, letter)
```

Output

```
0 a
1 e
2 i
3 o
4 u
```

Python Functions

Python Functions are a collection of statements that serve a specific purpose. The idea is to bring together some often or repeatedly performed actions and construct a function so that we can reuse the code included in it rather than writing the same code for different inputs over and over.

```
# A simple Python function
def fun():
    print("Welcome to GFG")

# Driver code to call a function
fun()
```

Output

```
Welcome to GFG
```

Function Arguments

Arguments are the values given between the function's parenthesis. A function can take as many parameters as it wants, separated by commas.

```
    if (x % 2 == 0):
        print("even")
    else:
        print("odd")


# Driver code to call the function
evenOdd(2)
evenOdd(3)
```

Output

```
even
odd
```

Return Statement in Python Function

The function return statement is used to terminate a function and return to the function caller with the provided value or data item.

```
# Python program to
# demonstrate return statement
def add(a, b):

    # returning sum of a and b
    return a + b

def is_true(a):

    # returning boolean of a
    return bool(a)

# calling function
res = add(2, 3)
print("Result of add function is {}".format(res))
```

Output

```
Result of add function is 5

Result of is_true function is True
```

The range() function

The Python range() function returns a sequence of numbers, in a given range.

```
# print first 5 integers
# using python range() function
for i in range(5):
    print(i, end=" ")
print()
```

Output

```
0 1 2 3 4
```

*args and **kwargs in Python

The *args and **kwargs keywords allow functions to take variable-length parameters. The number of non-keyworded arguments and the action that can be performed on the tuple are specified by the *args.**kwargs, on the other hand, pass a variable number of keyword arguments dictionary to function, which can then do dictionary operations.

```
def myFun(arg1, arg2, arg3):
    print("arg1:", arg1)
    print("arg2:", arg2)
    print("arg3:", arg3)
```

```
# pass arguments to this function :
args = ("Geeks", "for", "Geeks")
myFun(*args)

kwargs = {"arg1": "Geeks", "arg2": "for", "arg3": "Geeks"}
myFun(**kwargs)
```

Output

```
arg1: Geeks
arg2: for
arg3: Geeks
arg1: Geeks
arg2: for
arg3: Geeks
```

Interesting Facts about Functions in Python:

1. One can return multiple values in Python: In Python, we can return multiple values from a function. Python allows us to return multiple values by separating them with commas. These values are implicitly packed into a tuple and when the function is called, the tuple is returned

```
def func():
    return 1, 2, 3, 4, 5

one, two, three, four, five = func()
print(one, two, three, four, five)
```

Output

```
1 2 3 4 5
```

Data Structures in Python

List in Python

Python list is a sequence data type that is used to store the collection of data. Tuples and String are other types of sequence data types.

```
Var = ["Geeks", "for", "Geeks"]
print(Var)
```

Output

```
['Geeks', 'for', 'Geeks']
```

List comprehension

A Python list comprehension is made up of brackets carrying the expression, which is run for each element, as well as the for loop, which is used to iterate over the Python list's elements.

Also, Read - Python Array

```
# Using list comprehension to iterate through loop
List = [character for character in [1, 2, 3]]

# Displaying list
print(List)
```

Output

```
[1, 2, 3]
```

Interesting Facts about Lists in Python

- **Lists Have Been in Python Since the Beginning:** When Guido van Rossum created Python in the late 1980s, lists were an essential part of the language from the very first version (Python 1.0, released in 1991). Unlike languages like C or Java, where arrays have strict rules.
- **Inspired by ABC Language:** Python was influenced by the ABC

mutable (meaning they can be changed), which is one of the biggest reasons for their popularity.

Explore in detail about Interesting Facts About Python Lists

Dictionary in Python

A dictionary in Python is a collection of key values, used to store data values like a map, which, unlike other data types holds only a single value as an element.

```
Dict = {1: 'Geeks', 2: 'For', 3: 'Geeks'}
print(Dict)
```

Output

```
{1: 'Geeks', 2: 'For', 3: 'Geeks'}
```

Dictionary Comprehension

Like List Comprehension, Python allows dictionary comprehension. We can create dictionaries using simple expressions. A dictionary comprehension takes the form {key: value for (key, value) in iterable}

```
# Lists to represent keys and values
keys = ['a','b','c','d','e']
values = [1,2,3,4,5]

# but this line shows dict comprehension here
myDict = { k:v for (k,v) in zip(keys, values)}

# We can use below too
# myDict = dict(zip(keys, values))

print (myDict)
```

```
{'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}
```

Interesting Facts About Python Dictionary

- **Dictionaries Are Ordered:** Before Python 3.7, dictionaries did not preserve the order of insertion. However, with the introduction of Python 3.7 and beyond, dictionaries now maintain the order of the key-value pairs. Now we can rely on the insertion order of keys when iterating through dictionaries, which was previously a feature exclusive to **OrderedDict**.
- **Dictionaries Can Have Any Immutable Type as Keys:** The keys of a dictionary must be immutable, meaning they can be strings, numbers or tuples, but not lists or other mutable types. This feature ensures that the dictionary keys are hashable, maintaining the integrity and performance of lookups.

Explore in detail about Interesting Facts About Python Dictionary

Tuples in Python

Tuple is a list-like collection of Python objects. A tuple stores a succession of values of any kind, which are indexed by integers.

```
var = ("Geeks", "for", "Geeks")
print(var)
```

Output

```
('Geeks', 'for', 'Geeks')
```

Sets in Python

Python Set is an unordered collection of data types that can be iterated

```
var = {"Geeks", "for", "Geeks"}
print(var)
```

Output

```
{'for', 'Geeks'}
```

Python String

In Python, a string is a data structure that represents a collection of characters. A string cannot be changed once it has been formed because it is an immutable data type.

Creating and Accessing String

Strings in Python can be created using single quotes or double quotes or even triple quotes. In Python, individual characters of a String can be accessed by using the method of Indexing. Indexing allows negative address references to access characters from the back of the String.

```
String1 = "GeeksForGeeks"
print("Initial String: ")
print(String1)

# Printing First character
print("\nFirst character of String is: ")
print(String1[0])

# Printing Last character
print("\nLast character of String is: ")
print(String1[-1])
```

Output

```
Initial String:
```

```
Last character of String is:
s
```

String Slicing

Strings in Python can be constructed with single, double, or even triple quotes. The slicing method is used to access a single character or a range of characters in a String. A Slicing operator (colon) is used to slice a String.

```
# Creating a String
String1 = "GeeksForGeeks"
print("Initial String: ")
print(String1)

# Printing 3rd character
print("\nSlicing characters from 3-12: ")
print(String1[3])

# Printing characters between
# 3rd and 2nd last character
print("\nSlicing characters between " +
    "3rd and 2nd last character: ")
print(String1[3:-2])
```

Output

```
Initial String:
GeeksForGeeks

Slicing characters from 3-12:
k

Slicing characters between 3rd and 2nd last character:
ksForGee
```

Interesting Facts About Python strings

- Strings Can Be Concatenated Using the + Operator: We can easily combine (concatenate) strings using the + operator. This simple operation allows us to build longer strings from smaller ones, which is helpful when working with user input or constructing strings dynamically.

Explore in detail about Interesting Facts about Python Strings

Explore in Detail Python Data Structures and Algorithms

Python Built-In Function

There are numerous built-in methods in Python that make creating code easier. Learn about the built-in functions of Python in this post as we examine their numerous uses and highlight a few of the most popular ones.

For More Read, refer Python Built in Functions

Python OOPs Concepts

Object-oriented Programming (OOPs) is a programming paradigm in Python that employs objects and classes. It seeks to include real-world entities such as inheritance, polymorphisms, encapsulation and so on into programming. The primary idea behind OOPs is to join the data and the functions that act on it as a single unit so that no other portion of the code can access it.

- Class And Objects
- Polymorphism
- Encapsulation
- Inheritance
- Data Abstraction

In this example, we have a Car class with characteristics that represent the car's make, model and year. The _make attribute is protected with a single underscore _. The __model attribute is marked as private with two underscores __. The year attribute is open to the public.

We can use the getter function get_make() to retrieve the protected attribute _make. We can use the setter method set_model() to edit the private attribute __model. Using the getter method get_model(), we may retrieve the changed private attribute __model. There are no restrictions on accessing the public attribute year. We manage the visibility and accessibility of class members by using encapsulation with private and protected properties, offering a level of data hiding and abstraction.

```
class Car:
    def __init__(self, make, model, year):
        self._make = make  # protected attribute
        self.__model = model  # private attribute
        self.year = year  # public attribute

    def get_make(self):
        return self._make

    def set_model(self, model):
        self.__model = model

    def get_model(self):
        return self.__model


my_car = Car("Toyota", "Corolla", 2022)

print(my_car.get_make())  # Accessing protected attribute
my_car.set_model("Camry")  # Modifying private attribute
print(my_car.get_model())  # Accessing modified private
attribute
print(my_car.year)  # Accessing public attribute
```

```
Toyota
Camry
2022
```

Interesting facts about OOPs in Python:

1. Python Supports Multiple Programming Paradigms: One of Python's key strengths is its flexibility. Python supports several programming paradigms, including:

- Object-Oriented Programming (OOP)
- Functional Programming
- Procedural Programming

2. Everything in Python Is an Object: In Python, everything is an object, including data types such as integers, strings and functions. This allows for object-oriented programming (OOP) principles to be applied across all aspects of the language, including the ability to define classes, inheritance and methods.

```
x = 5

# <class 'int'>
print(type(x))

def greet(name):
    return f"Hello, {name}"

# <class 'function'>
print(type(greet))
```

Output

```
<class 'int'>
<class 'function'>
```

We define a pattern using a regular expression to match email addresses. The pattern r"\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}\b" is a common pattern for matching email addresses. Using the re.search() function, the pattern is then found in the given text. If a match is found, we use the match object's group() method to extract and print the matched email. Otherwise, a message indicating that no email was found is displayed.

```
import re

# Text to search
text = "Hello, my email is example@example.com"

# Define a pattern to match email addresses
pattern = r"\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]
{2,}\b"

# Search for the pattern in the text
match = re.search(pattern, text)

# Check if a match is found
if match:
    email = match.group()
    print("Found email:", email)
else:
    print("No email found.")
```

Output

```
Found email: example@example.com
```

MetaCharacters are helpful, significant and will be used in module re functions, which helps us comprehend the analogy with RE. The list of metacharacters is shown below.

- \ sed to drop the special meaning of character following it.

- $ Matches the end.
- . Matches any character except newline.
- | Means OR (Matches with any of the characters separated by it.
- ? Matches zero or one occurrence.
- * Any number of occurrences (including 0 occurrences).
- + One or more occurrences.
- {} Indicate the number of occurrences of a preceding RegEx to match.
- () Enclose a group of RegEx.

For More RegEx example, refer to Python RegEx.

Exception Handling in Python

In Python, Try and except statements are used to catch and manage exceptions. Statements that can raise exceptions are kept inside the try clause and the statements that handle the exception are written inside except clause.

```
a = [1, 2, 3]
try:
    print ("Second element = %d" %(a[1]))

    # Throws error since there are only 3 elements in array
    print ("Fourth element = %d" %(a[3]))

except:
    print ("An error occurred")
```

Output

```
Second element = 2
An error occurred
```

Debugging in Python

effectively.

1. Using Print Statements

```
x = 10
y = 20
print("x:", x)
print("y:", y)
result = x + y
print("result:", result)
```

Output

```
x: 10
y: 20
result: 30
```

2. Using pdb (Python Debugger)

The pdb module provides an interactive debugging environment. We can set breakpoints, step through the code, inspect variables and more. To start debugging, insert pdb.set_trace() where we want to start the debugger.

```
import pdb

x = 5
y = 10
pdb.set_trace()  # Debugger starts here
result = x + y
print(result)
```

Output

```
> /home/repl/2b752c75-c2c1-44d7-b2bd-fa43a3072b3c/main.py(6)
<module>()
```

Once the program reaches pdb.set_trace(), it will enter the interactive mode where we can run various commands like:

- **n (next):** Execute the next line of code.
- **s (step):** Step into a function.
- **c (continue):** Continue execution until the next breakpoint.
- **q (quit):** Exit the debugger.

3. Using logging Module

The logging module is more advanced than print() statements and is useful for debugging in production environments. It allows us to log messages with different severity levels (DEBUG, INFO, WARNING, ERROR, CRITICAL).

```
import logging

# Set up logging configuration
logging.basicConfig(level=logging.DEBUG)

# Log different messages
logging.debug("This is a debug message")
logging.info("This is an info message")
logging.warning("This is a warning message")
logging.error("This is an error message")
logging.critical("This is a critical message")
```

Output

```
DEBUG:root:This is a debug message
INFO:root:This is an info message
WARNING:root:This is a warning message
ERROR:root:This is an error message
CRITICAL:root:This is a critical message
```

File Handling in Python

operate on files.

```
import os

def create_file(filename):
    try:
        with open(filename, 'w') as f:
            f.write('Hello, world!\n')
        print("File " + filename + " created successfully.")
    except IOError:
        print("Error: could not create file " + filename)

def read_file(filename):
    try:
        with open(filename, 'r') as f:
            contents = f.read()
            print(contents)
    except IOError:
        print("Error: could not read file " + filename)

def append_file(filename, text):
    try:
        with open(filename, 'a') as f:
            f.write(text)
        print("Text appended to file " + filename + "
successfully.")
    except IOError:
        print("Error: could not append to file " + filename)

def rename_file(filename, new_filename):
    try:
        os.rename(filename, new_filename)
        print("File " + filename + " renamed to " +
                  new_filename + " successfully.")
    except IOError:
        print("Error: could not rename file " + filename)

def delete_file(filename):
```

```
    except IOError:
        print("Error: could not delete file " + filename)


if __name__ == '__main__':
    filename = "example.txt"
    new_filename = "new_example.txt"

    create_file(filename)
    read_file(filename)
    append_file(filename, "This is some additional text.\n")
    read_file(filename)
    rename_file(filename, new_filename)
    read_file(new_filename)
    delete_file(new_filename)
```

Output

```
File example.txt created successfully.
Hello, world!

Text appended to file example.txt successfully.
Hello, world!
This is some additional text.

File example.txt renamed to new_example.txt successfu...
```

Reading and Writing Files

Python provides built-in functions to handle file operations such as reading from and writing to files. We can use the open() function to work with files.

1. Opening a File:

The open() function is used to open a file and returns a file object.

- **'r':** Read (default mode) - Opens the file for reading.
- **'w':** Write - Opens the file for writing (creates a new file or overwrites an existing one).
- **'a':** Append - Opens the file for appending data.
- **'rb', 'wb':** Read/Write in binary mode.

2. Reading from a File:

There are several methods to read the contents of a file:

- read()
- readline()
- readlines()

```
with open('example.txt', 'r') as file:
    # Using read()
    content = file.read()
    print(content)

    # Resetting the pointer to the beginning of the file
    file.seek(0)

    # Using readline()
    first_line = file.readline()
    print(first_line)

    # Resetting the pointer again
    file.seek(0)

    # Using readlines()
    lines = file.readlines()
    print(lines)
```

3. Writing to a File:

We can write data to a file using the `write()` or `writelines()` methods:

write(): Writes a string to the file.

```
with open('filename.txt', 'w') as file:
    file.write("Hello, World!")
```

```
file.writelines(['Hello\n', 'World\n'])
```

4. Closing the File:

It is good practice to close the file after operations using file.close(), but it is recommended to use the with statement as it automatically handles file closure.

```
file.close()
```

Memory Management in Python

Memory management in Python is handled by the Python memory manager. Python uses an automatic memory management system, which includes garbage collection to reclaim unused memory and reference counting to track memory usage.

Key Concepts:

1. **Reference Counting**: Every object in Python has a reference count. When the count reaches zero, the object is deleted automatically.
2. **Garbage Collection**: Python uses a garbage collector to clean up circular references (when objects reference each other in a cycle).

```
import sys
x = 10
y = [1, 2, 3]

# Checking memory size of variables
print("Memory size of x:", sys.getsizeof(x))
print("Memory size of y:", sys.getsizeof(y))

# Reference counting
a = [1, 2, 3]
b = a
print(sys.getrefcount(a))
```

3

Decorators in Python

Decorators are used to modifying the behavior of a function or a class. They are usually called before the definition of a function we want to decorate.

1. property Decorator (getter)

```
@property
def name(self):
    return self.__name
```

2. setter Decorator

It is used to set the property 'name'

```
@name.setter
def name(self, value):
    self.__name=value
```

3. deleter Decorator

It is used to delete the property 'name'

```
@name.deleter
def name(self, value):
    print('Deleting..')
    del self.__name
```

Libraries in Python

Libraries in Python are collections of pre-written code that allow us to perform common tasks without having to write everything from scratch. They provide functions and classes to help with tasks like data

Basic Libraries

These libraries are part of Python's standard library and come bundled with Python installation. Some Basic Libraries are:

1. **math:** Provides mathematical functions such as trigonometric, logarithmic and basic arithmetic operations.
2. **datetime:** Works with dates and times, allows for formatting and manipulation of date/time data.
3. **os:** Provides functions for interacting with the operating system, such as working with files and directories.
4. **sys:** Provides access to system-specific parameters and functions, such as the interpreter's version or command-line arguments.

Data Science and Analysis Libraries

These libraries are essential for data manipulation, analysis and visualization. Some Data Science and Analysis Libraries are:

1. **numpy:** A fundamental package for scientific computing with Python. It provides support for large multidimensional arrays and matrices.
2. **pandas:** Used for data manipulation and analysis. It provides data structures like DataFrames to handle structured data.
3. **matplotlib:** A 2D plotting library to create static, animated and interactive visualizations.

Web Development Libraries

These libraries help us build and interact with web applications and services. Some Web Development Libraries are:

1. **flask:** A lightweight web framework for building web applications.
2. **requests:** A simple HTTP library for sending HTTP requests, handling

These libraries help build machine learning models, perform deep learning and work with AI algorithms. Some Machine Learning and AI Libraries are:

1. **scikit-learn:** A machine learning library that provides simple and efficient tools for data mining and data analysis.
2. **tensorflow:** A deep learning framework developed by Google, used for building neural networks and training AI models.

Python Modules

A library is a group of modules that collectively address a certain set of requirements or applications. A module is a file (.py file) that includes functions, class defining statements and variables linked to a certain activity. The term "standard library modules" refers to the pre-installed Python modules.

- Itertools
- Json
- Os
- Pathlib
- Random
- Shelve
- Zipfile

Interesting Facts about Python Modules:

1. import this: There is actually a poem written by Tim Peters named as THE ZEN OF PYTHON which can be read by just writing import this in the interpreter.

```
import this
```

Output

```
The Zen of Python, by Tim Peters
```

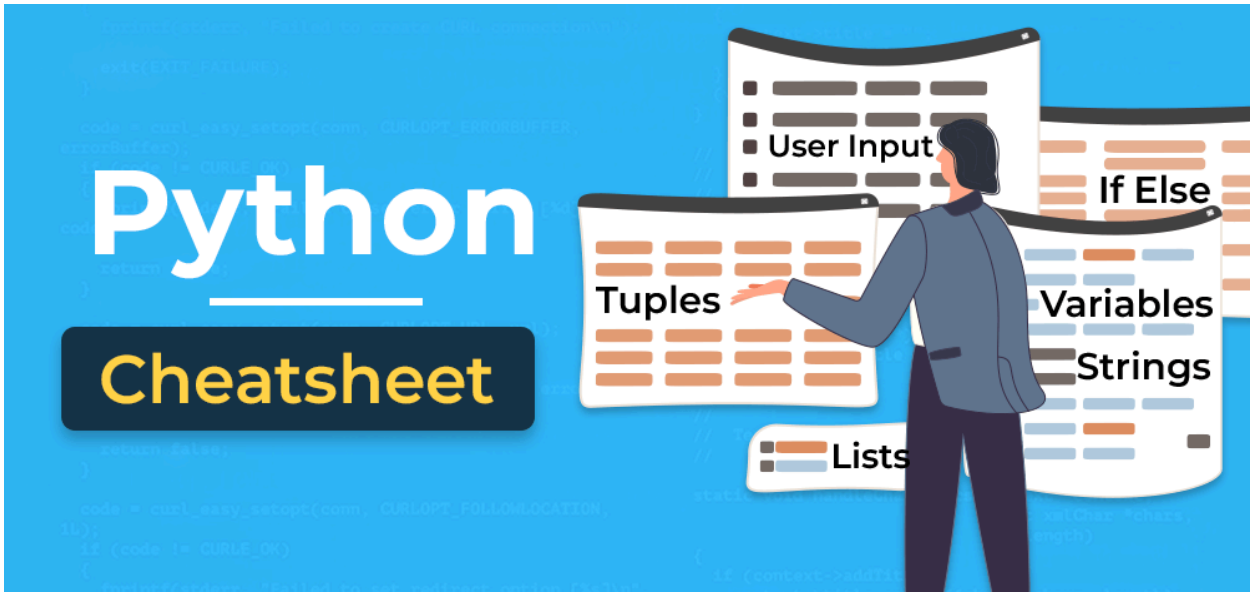
```
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.Errors should never pass
silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to
do it.
Although that way may not be obvious at first unless you're
Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good
idea.
Namespaces are one honking great idea -- let's do more of
those!
```

2. The "antigravity" Easter Egg in Python: The statement import antigravity in Python is a fun Easter egg that was added to the language as a joke. When we run this command, it opens a web browser to a comic from the popular webcomic xkcd (specifically, xkcd #353: "Python").

```
import antigravity
```

Python Interview Questions Answers

- Python Interview Questions Answers



Python Cheat Sheet

Comment More info

Campus Training Program

Similar Reads

1. Free Python Course Online [2025]
2. Python Crash Course
3. Python 3.13 New Features
4. 10 Best Python IDEs To Use [2025]
5. Top 5 Easter Eggs in Python
6. Python 3 basics
7. Latest Update on Python 4.0
8. 6 Best IDE's For Python in 2022
9. Best Python Bootcamp to Join in 2025
10. 10 Best Online Python Courses with Certificates [2025]

GeeksforGeeks
Sanchhaya Education Private Limited

Corporate & Communications Address:

A-143, 7th Floor, Sovereign Corporate Tower, Sector- 136, Noida, Uttar Pradesh (201305)

Registered Address:

K 061, Tower K, Gulshan Vivante Apartment, Sector 137, Noida, Gautam Buddh Nagar, Uttar Pradesh, 201305

NLP
Deep Learning

NodeJs
Bootstrap
Tailwind CSS

Python Tutorial

Python Examples
Django Tutorial
Python Projects
Python Tkinter
Web Scraping
OpenCV Tutorial
Python Interview Question

Computer Science

GATE CS Notes
Operating Systems
Computer Network
Database Management System
Software Engineering
Digital Logic Design
Engineering Maths

DevOps

Git
AWS
Docker
Kubernetes
Azure
GCP
DevOps Roadmap

System Design

High Level Design
Low Level Design
UML Diagrams
Interview Guide
Design Patterns
OOAD
System Design Bootcamp
Interview Questions

School Subjects

Mathematics
Physics
Chemistry
Biology
Social Science
English Grammar

Databases

SQL
MYSQL
PostgreSQL
PL/SQL
MongoDB

Preparation Corner

Company-Wise Recruitment Process
Aptitude Preparation
Puzzles
Company-Wise Preparation

More Tutorials

Software Development
Software Testing
Product Management
Project Management
Linux
Excel
All Cheat Sheets

Courses

IBM Certification Courses
DSA and Placements

Programming Languages

C Programming with Data Structures
C++ Programming Course

DevOps & Cloud

Clouds/Devops

DevOps Engineering

AWS Solutions Architect Certification

Salesforce Certified Administrator Course

GATE 2026

GATE CS Rank Booster

GATE DA Rank Booster

GATE CS & IT Course - 2026

GATE DA Course 2026

GATE Rank Predictor