

## Python Fundamental

1. What is Python, and why is it popular for programming?
2. What are the main features of Python that make it beginner-friendly?
3. How do you write and run a Python program?
4. Explain Python's dynamic typing and give an example.
5. What is the role of indentation in Python, and how does it differ from other languages?
6. What is an if statement, and how is it used in Python?
7. Explain the syntax of an if-elif-else structure.
8. What is the difference between = and == operators in Python?
9. How does the elif statement work in Python? Provide an example.
10. Explain the use of logical operators (and, or, not) in if statements.
11. What is a for loop in Python, and how does it work?
12. How does the range() function work in Python? Give examples.
13. Explain how to use a for loop to iterate over a list.
14. How can you use a for loop with the enumerate() function, and why is it useful?
15. What is list comprehension, and how is it different from a for loop?
16. What is a while loop, and when would you use it over a for loop?
17. Explain the syntax of a while loop and give an example.
18. What is an infinite loop, and how can it occur with a while loop?
19. How can you use break and continue statements in a while loop?
20. What is the difference between for and while loops, and when would you use each?

## String

1. What is a string in Python, and how do you create one?
2. Explain the immutability of strings in Python. What does it mean, and why is it important?
3. How do you access individual characters in a string?
4. What is string slicing, and how do you perform it in Python?
5. How can you concatenate two or more strings in Python?
6. What are some common string methods in Python, and what do they do?
7. How does the len() function work with strings?
8. Explain the difference between upper(), lower(), and title() string methods.
9. How do you remove whitespace from the beginning or end of a string?
10. What is string formatting, and what are the different ways to format strings in Python?
11. Explain f-strings (formatted string literals) in Python and give an example.
12. How do you check if a substring exists within a string in Python?
13. What is the purpose of the find() and index() methods? How do they differ?
14. How can you split a string into a list of substrings?
15. How do you join a list of strings into a single string in Python?
16. Explain how to replace a substring within a string using the replace() method.
17. What is string interpolation, and how is it achieved in Python?
18. How can you reverse a string in Python?
19. How do startswith() and endswith() methods work with strings?
20. How do you compare two strings in Python? What operators or functions can be used?

## List

1. What is a list in Python, and how do you create one?
2. Explain the difference between lists and arrays in Python.
3. How do you access elements in a list by index?
4. How can you add elements to a list in Python?
5. Explain the difference between append() and extend() methods.
6. How do you insert an element at a specific position in a list?

7. How can you remove an element from a list? Describe `remove()`, `pop()`, and `del`.
8. What is list slicing, and how do you perform it?
9. How do you iterate over elements in a list using a `for` loop?
10. What is a list comprehension, and how is it used?
11. How do you find the length of a list in Python?
12. Explain how you would check if an item exists within a list.
13. How do you sort a list in Python? Explain `sort()` vs `sorted()`.
14. What is the difference between shallow copy and deep copy of a list?
15. How can you reverse the elements of a list?
16. How do you count occurrences of an element in a list?
17. How do you find the index of a particular element in a list?
18. Explain how the `min()` and `max()` functions work with lists.
19. How do you merge or concatenate two lists in Python?
20. What is the difference between `copy()` and using `[:] for list duplication?`

## Tuple

1. What is a tuple in Python, and how do you create one?
2. How is a tuple different from a list in Python?
3. What does it mean for a tuple to be immutable?
4. How can you access elements in a tuple by index?
5. Explain how tuple slicing works in Python.
6. How can you create a single-element tuple?
7. What are some situations where using a tuple is more beneficial than a list?
8. How can you concatenate two tuples in Python?
9. Can you modify a tuple in Python? If not, why?
10. How do you unpack values from a tuple into individual variables?
11. What is tuple packing and unpacking? Give an example.
12. How can you check if an element exists within a tuple?
13. How do you find the length of a tuple in Python?
14. Explain the use of `count()` and `index()` methods in tuples.
15. How do you iterate over elements in a tuple?
16. Can a tuple contain mutable data types, such as lists? If yes, how does it affect immutability?
17. How do you convert a list to a tuple, and vice versa?
18. How does Python handle memory management with tuples compared to lists?
19. What are named tuples, and how do they differ from regular tuples?
20. Explain how to use a tuple as a dictionary key.

## Dictionary

1. What is a dictionary in Python, and how do you create one?
2. How do dictionaries differ from lists and tuples in Python?
3. Explain the concept of key-value pairs in dictionaries.
4. How do you access the value associated with a specific key in a dictionary?
5. How can you add a new key-value pair to an existing dictionary?
6. How do you update the value of an existing key in a dictionary?
7. Explain the difference between `get()` and direct key access in a dictionary.
8. How can you remove a key-value pair from a dictionary? Describe `pop()`, `popitem()`, and `del`.
9. What are dictionary methods like `keys()`, `values()`, and `items()` used for?
10. How can you iterate over key-value pairs in a dictionary?
11. What happens if you try to access a key that doesn't exist in a dictionary?
12. How do you check if a key exists in a dictionary?
13. What is a dictionary comprehension, and how is it used?
14. Explain how to merge two dictionaries in Python.

15. Can dictionary keys be mutable? Why or why not?
16. What are some use cases for using dictionaries in Python?
17. How do you sort a dictionary by keys or by values?
18. What is the purpose of the fromkeys() method in dictionaries?
19. How do clear() and copy() methods work in dictionaries?
20. Explain how a dictionary can be used to implement a simple lookup table.

## User Defined Function

1. What is a function in Python, and why are functions used?
2. How do you define a user-defined function in Python? Provide an example.
3. What is the purpose of the return statement in a function?
4. What is the difference between a function that returns a value and one that doesn't?
5. How do you call a function in Python?
6. What are parameters and arguments in the context of functions?
7. What is the difference between positional and keyword arguments?
8. Explain default parameters in Python functions and give an example.
9. What is the purpose of \*args in a function definition?
10. How does \*\*kwargs work in a function, and when would you use it?
11. Can you explain the scope of variables inside a function (local vs. global scope)?
12. What is the global keyword, and when is it used in a function?
13. What does the nonlocal keyword do, and how is it different from global?
14. How can you return multiple values from a function in Python?
15. What is a lambda function, and how does it differ from a standard function?
16. Explain the concept of a recursive function and provide a simple example.
17. What are higher-order functions, and how can they be used in Python?
18. What is function overloading, and does Python support it?
19. Explain the concept of function decorators and give an example of how to use one.
20. How can you use functions to improve code modularity and reusability?

## Python Object Oriented Programming

1. What is Object-Oriented Programming (OOP), and why is it used?
2. Explain the four main principles of OOP: Encapsulation, Abstraction, Inheritance, and Polymorphism.
3. What is a class in Python, and how do you define one?
4. What is an object, and how is it related to a class?
5. What is the purpose of the self keyword in Python classes?
6. How do you create an instance (object) of a class in Python?
7. Explain the difference between instance variables and class variables.
8. What is the purpose of the \_\_init\_\_ method in Python?
9. What is encapsulation, and how is it implemented in Python?
10. How do you make an attribute or method private in Python?
11. What is abstraction, and how does it differ from encapsulation?
12. How can you achieve abstraction in Python?
13. What are getter and setter methods, and how are they used in Python?
14. Explain the @property decorator in Python and its purpose.
15. How do public, protected, and private access modifiers work in Python?
16. What is inheritance, and how does it work in Python?
17. Explain the difference between single, multiple, and multilevel inheritance with examples.
18. How do you create a subclass in Python?
19. What is the super() function, and when would you use it?
20. What is method overriding, and how is it implemented in Python?
21. What are the potential issues with multiple inheritance, and how does Python handle them?
22. What is the MRO (Method Resolution Order) in Python, and how does it affect

inheritance?

23. What is polymorphism in OOP, and how does it work in Python?
24. What is method overloading, and does Python support it?
25. Explain operator overloading and give an example in Python.
26. How do the `_str_` and `_repr_` methods work in Python classes?
27. What is duck typing, and how does it apply to Python?
28. How is polymorphism useful in designing flexible and maintainable code?
29. What are abstract classes and interfaces, and how are they implemented in Python?
30. What is composition, and how is it different from inheritance?

## Django Framework

1. What is Django, and what are its main features?
2. What is the MTV (Model-Template-View) architecture in Django?
3. How do you create a new Django project, and how is it structured?
4. What is a Django app, and how is it different from a project?
5. What is the purpose of `settings.py` file in Django?
6. How do you define a model in Django, and what are some common field types?
7. What is Django ORM, and how does it simplify database interactions?
8. How do you create and apply database migrations in Django?
9. Explain the purpose of `makemigrations` and `migrate` commands.
10. What are Django views, and what is the difference between function-based views (FBVs) and class-based views (CBVs)?
11. How do URL patterns work in Django, and how do you define a URL in `urls.py`?
12. What are templates in Django, and how do they help with the presentation layer?
13. How does Django's template language handle variables, filters, and tags?
14. What is a Django form, and how does it simplify form handling?
15. How does Django handle user authentication?
16. Explain middleware in Django and give examples of common middleware types.
17. How does Django's built-in admin interface work, and how do you customize it?
18. What are Django signals, and how can they be used?
19. How do you manage static and media files in Django?
20. What are some common security features provided by Django?

## Django Rest Framework

1. What is Django REST Framework (DRF), and why is it used?
2. What is a serializer in DRF, and how does it differ from Django forms?
3. Explain how to create a basic API view in DRF using function-based views (FBVs).
4. What are class-based views (CBVs) in DRF, and how are they used for building APIs?
5. How does the `APIView` class work in DRF, and what are its key features?
6. What are model serializers, and how do they simplify API development?
7. How does authentication work in DRF, and what types of authentication does it support?
8. What is pagination in DRF, and how do you enable it for an API?
9. How does DRF handle permissions, and what are some common permission classes?
10. What is a `ViewSet` in DRF, and how does it simplify routing and view handling?