```
1.What is Python? Mention two of its key features.
Python is a high-level, interpreted programming language known for its simplicity and readability.
It is widely used in web development, data analysis, artificial intelligence, automation, and more.
features-
Easy-to-Read Syntax
Interpreted Language
Dynamically Typed
object oriented.

2.How do you write a single-line comment in Python?
hash symbol (#)

3.How do you write a multi-line comment in Python?
Using triple quotes (''' or """)

4.What is the difference between print() and return?
print(): Used for debugging or showing results to the user
return: Used to store, reuse, or manipulate the result in your program

5.How do you get user input in Python?
input() function.

6.How do you check the version of Python installed?
python --version

7.Is Python case-sensitive? Give an example
Yes, Python is case-sensitive.
Ex = "myPet" and "mypet" are different

8.How do you run a Python script from the command line?
python script_name.py

9.What are keywords in Python? How can you list them?
if, else, elif
for, while, break, continue
def, return, class
import, from, as
True, False, None
try, except, finally

10.How do you declare a variable in Python?
variable_name = value

11.What’s the difference between = and ==?
= (Single Equals Sign)-
Assignment Operator
Used to assign a value to a variable.

== (Double Equals Sign)
Comparison Operator
Used to check if two values are equal.
Returns True or False (Boolean).

12.How do you swap two variables without using a third variable?
a = 5
b = 10
a, b = b, a  # Swapping
print("a:", a)
print("b:", b)

13.How do you write a one-line if statement?
The ternary operator is a concise way to write an if-else statement in one line.
synatx - value_if_true if condition else value_if_false

14.What’s the difference between None and 0 in Python?
Represents the absence of a value or null.
```

```
Type: NoneType

A number – specifically, the integer zero.
Type: int

15.What is indentation in Python and why is it important?
Indentation is used to define blocks of code instead of braces {} like in many other languages.

16.What are Python’s built-in data types?
Python has several built-in data types,
Numeric types: int, float, complex
Sequence types: list, tuple, range
Text type: str
Set types: set, frozenset
Mapping type: dict
Boolean type: bool
Binary types: bytes, bytearray, memoryview
NoneType: None

17.How do you check the type of a variable?
Use the built-in function type()

18.What’s the difference between a list and a tuple?
list are mutable
Tuple are immutable

19.How do you create a dictionary in Python?
Using curly braces {} with key value pairs separated by colons:

20.What’s the difference between append() and extend() for lists?
append() adds a single element to the end ofthe list
extend() adds all elements from an iterable (like another list) to the end.

21.How do you remove an item from a list?
remove()
pop()

22.How do you reverse a list in Python?
reverse()

23.How do you sort a list in ascending order?
Use the sort() method (in-place) or the sorted() function (returns a new sorted list)

24.What is the difference between shallow copy and deep copy?
Shallow copy copies the outer object, but inner objects are referenced.
Deep copy copies everything recursively (outer and inner objects).

25.How do you convert a string to lowercase?
s = "Hello"
print(s.lower())

26.How do you check if a string starts with a particular word?
s = "Hello world"
print(s.startswith("Hello"))

27.What’s the difference between is and ==?
== checks value equality.
is checks object identity (whether two variables point to the same object).

28.How do you merge two dictionaries in Python 3.9+?
Use the merge operator |.

29.How do you find the length of a dictionary?
Use len()

30.How do you create a set?
Using set() or curly braces {} with element
```

```
31.What’s the difference between set() and {} in Python?
{} creates an empty dictionary, not a set.
Use set() to create an empty set

32.How do you find the union of two sets?
Use .union() or | operator

33.How do you find the intersection of two sets?
Use .intersection() or & operator

34.What’s the difference between remove() and discard() in sets?
remove() raises a KeyError if the element is not found.
discard() does nothing if the element is not found.

35.How do you convert a list into a tuple?
Use the tuple() function

36.How does the if-elif-else structure work in Python?
x = 10
if x > 10:
    print("Greater than 10")
elif x == 10:
    print("Equal to 10")
else:
    print("Less than 10")

37.What is the difference between for and while loops?
For loop is used to iterate over a sequence of items.
While loop is used to repeatedly execute a block of statements while a condition is true.
For loops are designed for iterating over a sequence of items. Eg. list, tuple, etc.
While loop is used when the number of iterations is not known in advance or when we want to repeat a block of code until a
For loop require a sequence to iterate over.
While the loop requires an initial condition that is tested at the beginning of the loop.
For loop is typically used for iterating over a fixed sequence of items
While loop is used for more complex control flow situations.

38.How do you loop through a dictionary’s keys and values?
my_dict = {'a': 1, 'b': 2}
for key, value in my_dict.items():
    print(key, value)

39.How do you break out of a loop?
for i in range(5):
    if i == 3:
        break
    print(i)  # Output: 0, 1, 2

40.How do you skip the current iteration in a loop?
for i in range(5):
    if i == 2:
        continue
    print(i)  # Output: 0, 1, 3, 4

41.What is an infinite loop? Give an example.
An infinite loop runs forever (unless stopped with break, error, or interruption):
Ex-
while True:
    print("This will run forever")

42.How do you use the range() function?
range(stop)               # from 0 to stop-1
range(start, stop)        # from start to stop-1
range(start, stop, step)

43.What’s the difference between range(5) and range(1,5)?
range(5)  0, 1, 2, 3, 4
range(1, 5) 1, 2, 3, 4
```

range(5) starts from 0.
range(1, 5) starts from 1.

44.How do you iterate over both index and value in a list?
Use enumerate():
Ex-

```
fruits = ['apple', 'banana', 'cherry']
for index, fruit in enumerate(fruits):
    print(index, fruit)
```

45.How does the else clause work with loops in Python?
The else block in a loop runs only if the loop completes without hitting break

```
for i in range(3):
    print(i)
else:
    print("Loop completed successfully")
# Example with break
    for i in range(3):
    if i == 1:
        break
    print(i)
else:
    print("Won't run because of break")
```

46.How do you use a nested loop?
A nested loop is a loop inside another loop.
Ex-

```
for i in range(2):
    for j in range(3):
        print(f"i={i}, j={j}")
```

47.How do you loop through multiple lists simultaneously?
Use the built-in zip() function.

```
names = ['Alice', 'Bob', 'Charlie']
scores = [85, 90, 95]

for name, score in zip(names, scores):
    print(f"{name}: {score}")
```

48.How do you reverse iterate over a list?
Use the built-in reversed() function or slicing.

```
lst = [1, 2, 3, 4]

# Using reversed()
for item in reversed(lst):
    print(item)

# Using slicing
for item in lst[::-1]:
    print(item)
```

49.What is a list comprehension? Give an example.
A list comprehension is a concise way to create lists in a single line of code.

Example:

```
squares = [x**2 for x in range(5)]
print(squares)  # Output: [0, 1, 4, 9, 16]
```

It’s the same as:

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

50. How do you use a conditional inside a list comprehension?
There are two ways:
(a) Filter condition (if at the end) – keeps only matching values:

```
even_numbers = [x for x in range(10) if x % 2 == 0]
even_numbers = [x for x in range(10) if x % 2 == 0]
print(even_numbers)  # [0, 2, 4, 6, 8]

(b) Conditional expression (if-else inside) – changes the output:
labels = ['even' if x % 2 == 0 else 'odd' for x in range(5)]
print(labels)  # ['even', 'odd', 'even', 'odd', 'even']

51.How do you define a function in Python?
Use the def keyword, followed by the function name and parentheses.
Example:
def greet(name):
    print(f"Hello, {name}!")

52.What is the difference between positional and keyword arguments?
Positional arguments are passed based on order:
def greet(name, age):
    print(name, age)

greet("Alice", 30)  # Positional
Keyword arguments are passed using the parameter name:
greet(age=30, name="Alice")  # Keyword

53.What are default parameter values in functions?
These provide default values if arguments aren't passed:
def greet(name="Guest"):
    print(f"Hello, {name}!")

54.How do you pass a variable number of arguments to a function?
*args for any number of positional arguments:
def add(*numbers):
    return sum(numbers)
**kwargs for any number of keyword arguments:
def describe(**info):
    print(info)

55.How do you return multiple values from a function?
Use a tuple (implicit or explicit):

def get_person():
    return "Alllu", 30

name, age = get_person()

56.What is a lambda function? Give an example.
An anonymous function defined with lambda:
square = lambda x: x * x
print(square(5))  # Output: 25

57.What's the difference between local and global variables?
Local variable: Defined inside a function, only accessible there.
Global variable: Defined outside functions, accessible anywhere.

58.How do you modify a global variable inside a function?
Use the global keyword:
count = 0
def increment():
    global count
    count += 1

59.What is recursion in Python? Give an example.
A function calling itself:
def factorial(n):
    if n == 0:
        return 1
    return n * factorial(n - 1)

60.What is a docstring in Python functions?
A string literal used to document a function:
```

```
def greet(name):
    """Prints a greeting to the user."""
    print(f"Hello, {name}!")
Access with help(greet) or greet.__doc__.

61.How do you use type hints in functions?
Add expected types for parameters and return values:
def add(x: int, y: int) -> int:
    return x + y

62.What are function annotations?
Metadata about parameters and return types. Type hints are a common use:
def greet(name: str) -> None:
    print(f"Hello, {name}!")
    You can access annotations via function.__annotations__.

63.What is the purpose of the pass statement?
A placeholder that does nothing. Used when a block is syntactically required:
def todo():
    pass  # TODO: implement later

64.How do you define a class in Python?
Use the class keyword:
class Person:
    def greet(self):
        print("Hello!")

65.How do you create an object from a class?
Call the class like a function:
p = Person()  # creates an object of class Person
p.greet()     # calls the method on the object

66.What is __init__ in Python?
__init__ is a special method called when an object is created (constructor). It initializes the object’s attributes.

class Person:
    def __init__(self, name):
        self.name = name

    def greet(self):
        print(f"Hello, I'm {self.name}")

p = Person("Allu")
p.greet()  # Output: Hello, I'm Allu

67.What is the difference between instance variables and class variables?
Instance variables: Unique to each object, defined using self.
Class variables: Shared among all instances of a class.
class Dog:
    species = "Canine"  # class variable

    def __init__(self, name):
        self.name = name  # instance variable

dog1 = Dog("Buddy")
dog2 = Dog("Max")

print(dog1.name)     # Buddy
print(dog2.name)     # Max
print(dog1.species)  # Canine

68.What is inheritance in Python?
Inheritance allows a class to derive attributes and methods from another class.
class Animal:
    def speak(self):
        print("Animal speaks")

class Dog(Animal):  # Dog inherits from Animal
```

```
        def bark(self):
            print("Dog barks")

    d = Dog()
    d.speak()  # Inherited method
    d.bark()   # Dog's own method

    69.How do you call a parent class constructor?
    Use super().__init__() in the child class constructor:
    class Parent:
        def __init__(self, name):
            self.name = name

    class Child(Parent):
        def __init__(self, name, age):
            super().__init__(name)  # calls Parent's constructor
            self.age = age

    70.What’s the difference between method overloading and overriding?
    Concept       Overloading                                      Overriding
    Definition    Same method name,                                 different parameters Child class redefines parent method
    Support       in Python Not directly supported
                  (use default args or *args)                                      Fully supported
    Ex-
                  class A:
        def greet(self):
            print("Hello from A")

    class B(A):
        def greet(self):  # Overrides A's method
            print("Hello from B")

    71.What is multiple inheritance?
    A class inherits from more than one parent class:

    class A:
        def method_a(self):
            print("A")

    class B:
        def method_b(self):
            print("B")

    class C(A, B):
        pass

    c = C()
    c.method_a()
    c.method_b()

    72.What’s the role of super() in Python?
    Calls methods from the parent class.
    Especially useful in inheritance to avoid hardcoding parent class names.
    Supports method resolution order (MRO) in multiple inheritance.
    super().__init__()

    73.What are magic methods in Python? Give two examples.
    pecial methods with double underscores (__dunder__) that define object behavior for built-in operations.
    Examples:
    __init__: Constructor
    __str__: String representation
    Others: __add__, __len__, __eq__, etc.

    74.What does __str__ do in a class?
    class Book:
        def __init__(self, title):
            self.title = title
        def __str__(self):
            return f"Book: {self.title}"

    b = Book("Python 101")
    print(b)  # Output: Book: Python 101

    75.What is polymorphism in Python?
    class Dog:
        def speak(self):
            return "Bark"

    class Cat:
        def speak(self):
            return "Meow"

    def make_sound(animal):
        print(animal.speak())

    make_sound(Dog())
    make_sound(Cat())

    76.What is encapsulation in Python?
    Restricting direct access to object data and methods to protect internal state.
    Achieved using private attributes and getters/setters.

    class Person:
        def __init__(self):
            self.__name = "Alice"  # private

        def get_name(self):
            return self.__name

    77.What are @staticmethod?
    A method that doesn't access self or cls. It behaves like a regular function but belongs to the class.
    class Math:
        @staticmethod
        def add(a, b):
            return a + b

    print(Math.add(2, 3))  # Output: 5

    78.How do you make an attribute private in Python?
    class Car:
        def __init__(self):
            self.__speed = 0  # private

        def get_speed(self):
            return self.__speed
    Internally, __speed becomes _Car__speed to prevent direct access.

    79.How do you import a module in Python?
    Use the import keyword:
    import math
    print(math.sqrt(16))  # Output: 4.0

    80.What’s the difference between import module and from module import?
    import module -Imports the whole module. You access functions with module.name.
    from module -import name  Imports only a specific part (function, class, variable).
    import math
    print(math.pi)
    from math import pi
    print(pi)

    81.How do you find the location of an imported module?
    Use the __file__ attribute:
    import math
    print(math.__file__)
    Note: Not all modules (like built-ins) have a __file__ attribute
```

Note: Not all modules (like built-ins) have a __file__ attribute.
You can also use inspect:

```
import inspect
import math
print(inspect.getfile(math))  # May raise error for built-in modules
```

82.How do you create your own module in Python?
Create a .py file (e.g., mymodule.py) with some functions or variables:

```
# mymodule.py
def greet(name):
    return f"Hello, {name}!"
```

Import and use it in another script:

```
import mymodule
print(mymodule.greet("Alice"))
```