```
#1. Write a function to add two numbers and return the result.

def add(a, b):
    return a + b
add(5,6)
```

```
11
```

```
#2. Write a function that takes a number and returns its square.
def square(n):
    return n * n
square(5)
```

```
25
```

```
#3. Write a function to check whether a number is even or odd.
def even_odd(n):
    return "Even" if n % 2 == 0 else "Odd"
even_odd(5)
```

```
'Odd'
```

```
#4. Write a function to find the maximum of three numbers.
def max_three(a, b, c):
    return max(a, b, c)

max_three(1,2,3)
```

```
3
```

```
#5.Write a function to calculate the factorial of a number (without recursion).
def factorial(n):
    f = 1
    for i in range(1, n+1):
        f *= i
    return f

factorial(5)
```

```
120
```

```
#6. Write a function to check whether a number is prime.

def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True
is_prime(5)
```

```
True
```

```
#7. Write a function to return the sum of all numbers in a given list.

def sum_list(l):
    return sum(l)

sum_list([1,2,3,4,5])
```

```
15
```

```
#8. Write a function to count the number of digits in a number.

def count_digits(n):
    return len(str(n))

count_digits(12345)
```

```
5
```

```
#9. Write a function to reverse a number.

def reverse_number(n):
```

```
    return int(str(n)[::-1])

reverse_number(12345)
```

```
54321
```

```
#10. Write a function to reverse a string.

def reverse_string(s):
    return s[::-1]

reverse_string("hello")
```

```
'olleh'
```

```
#11. Write a function to count vowels in a string.

def count_vowels(s):
    vowels = "aeiouAEIOU"
    count = 0
    for char in s:
        if char in vowels:
            count += 1
    return count

count_vowels("hello")
```

```
2
```

```
#12. Write a function that returns both the minimum and maximum from a list.

def min_max(l):
    return min(l), max(l)

min_max([1,2,3,4,5])
```

```
(1, 5)
```

```
#13. Write a function using default arguments to calculate simple interest.

def simple_interest(p, r, t):
    return (p * r * t) / 100

simple_interest(1000, 10, 2)
```

```
200.0
```

```
#14. Write a function that accepts any number of arguments and returns their sum using
# *args.
def sum_args(*args):
    return sum(args)

sum_args(1,2,3,4,5)
```

```
15
```

```
#15. Write a function that accepts key-value pairs using **kwargs and prints them in formatted
# form.
def print_kwargs(**kwargs):
    for k, v in kwargs.items():
        print(f"{k}: {v}")

print_kwargs(a=1, b=2, c=3)
```

```
a: 1
b: 2
c: 3
```

```
#16. Write a function to check whether a string is a palindrome.
def is_palindrome(s):
    return s == s[::-1]

is_palindrome("madam")
```

```
True
```

```
# 17. Write a function that returns the nth Fibonacci number using loops.
def fibonacci(n):
    a, b = 0, 1
    for _ in range(n):
        a, b = b, a+b
    return a
fibonacci(10)
```

```
55
```

```
#18. Write a function that swaps two numbers without using a third variable.
def swap(a, b):
    return b, a
swap(1,2)
```

```
(2, 1)
```

```
# 19. Write a function that returns whether a year is a leap year.
def leap_year(y):
    return y % 400 == 0 or (y % 4 == 0 and y % 100 != 0)

leap_year(2000)
```

```
True
```

```
#20. Write a function that converts Celsius to Fahrenheit.
def c_to_f(c):
    return (c * 9/5) + 32
c_to_f(0)
```

```
32.0
```

```
#21. Write a function that counts how many times a function is called.
count = 0
def call_counter():
    global count
    count += 1
    return count
call_counter()
```

```
1
```

```
#22. Write a function that returns another function.
def outer():
    def inner():
        return "Hello"
    return inner
outer()()
```

```
'Hello'
```

```
#23. Write a function that takes another function as an argument and executes it.
def execute(func):
    func()

def hello():
    print("Hello")

execute(hello)
```

```
Hello
```

```
#24. Write a lambda function to find the square of a number.
square = lambda x: x*x
square(5)
```

```
25
```

```
#25. Write a lambda function to find the maximum of two numbers.
maximum = lambda a, b: a if a > b else b
```

```
maximum(5,6)
```

```
6
```

```
#26. Write a function to calculate the power of a number (a^b).
def power(a, b):
    return a ** b
power(2,3)
```

```
8
```

```
#27. Write a function to find the sum of digits of a number.
def sum_digits(n):
    return sum(int(d) for d in str(abs(n)))

sum_digits(12345)
```

```
15
```

```
#28. Write a function that checks whether all arguments passed are positive numbers.
def all_positive(*args):
    return all(x > 0 for x in args)

all_positive(1,2,3,4,5)
```

```
True
```

```
#29. Write a function that returns multiple values (sum, difference, product) of two numbers.
def calc(a, b):
    return a+b, a-b, a*b
calc(5,6)
```

```
(11, -1, 30)
```

```
#30. Write a function that prints a pattern using only function calls.
def star():
    print("*")

def pattern():
    for _ in range(5):
        star()
        for _ in range(4):
            print("*", end="")
        print()
pattern()
```

```
*
****
*
****
*
****
*
****
*
****
```

```
#31. Write a function to calculate the average of numbers using *args.
def average(*args):
    return sum(args) / len(args)

average(1,2,3,4,5)
```

```
3.0
```

```
#32. Write a function to merge two dictionaries using **kwargs.
def merge_dicts(**kwargs):
    return kwargs

merge_dicts(a=1, b=2, c=3)
```

```
{'a': 1, 'b': 2, 'c': 3}
```

```
#33. Write a function that demonstrates local and global variable behavior.
x = 10
def demo():
    global x
    x = 20

demo()
print(x)
```

```
20
```

```
#34. Write a function that accepts another function and applies it to a value.
def apply(func, val):
    return func(val)

def square(x):
    return x*x

apply(square, 5)
```

```
25
```

```
# 35. Write a function to generate a multiplication table of a number.
def table(n):
    for i in range(1, 11):
        print(n*i)
table(6)
```

```
6
12
18
24
30
36
42
48
54
60
```

```
# 36. Write a function to check whether a number is Armstrong or not.
def is_armstrong(n):
    return n == sum(int(d)**len(str(n)) for d in str(n))

is_armstrong(153)
```

```
True
```

```
# 37. Write a function that validates a password based on given rules.
def valid_password(p):
    return len(p)>=8 and any(c.isdigit() for c in p)

valid_password("hello123")
```

```
True
```

```
# 38. Write a function to count words in a sentence.
def word_count(s):
    return len(s.split())

word_count("hello world")
```

```
2
```

```
# 39. Write a function to remove duplicates from a list.
def remove_duplicates(lst):
    return list(set(lst))

remove_duplicates([1,2,3,4,5,1,2,3])
```

```
[1, 2, 3, 4, 5]
```

```
# 40. Write a function to check whether two strings are anagrams.
def is_anagram(a, b):
    return sorted(a) == sorted(b)

is_anagram("hello", "olleh")
```

```
True
```

```
# 41. Why is Python called a high-level language?
A high-level programming language is a language that is:
-- Close to human language
-- Easy to read, write, and understand
-- Abstracted away from machine hardware
```

```
# 42. What are the key features of Python?
-- Simple, interpreted, portable, OOP, dynamic typing.
```

```
# 43. Is Python case-sensitive?
-- yes, Python is case-sensitive.
age = 10
Age = 20
print(age)  # 10
print(Age)  # 20
```

```
10
20
```

```
# 44. What is the difference between Python 2 and Python 3?
| Python 2         | Python 3           |
| ---------------- | ------------------ |
| print statement  | print() function   |
| ASCII by default | Unicode by default |
| Deprecated       | Actively used      |
```

```
# 45. What is an interpreter?
An interpreter executes a code line by line.

Example->

If error occurs at line 5, lines 1–4 already execute.
```

```
# 46. How does Python execute code?
Source code → Bytecode (.py)

Bytecode → Python Virtual Machine (PVM)

Output produced
```

```
# 47. What is dynamic typing?
Variable type is decided at runtime.

Example ->
x = 10
x = "Python"
```

```
# 48. What is strong typing in Python?
Python does not allow mixing incompatible types.
"5" + 5   #  Error
```

```
# 49. What is a variable?
A variable is a name that refers to a value in memory.
```

```
# 50. How are variables created in Python?
By assignment.
x = 100
y = "Hello"
```

```
#51. What are naming rules for variables?
* Letters, digits, underscore
* Cannot start with digit
* No keywords
```

```
* Case sensitive
valid_name = 10
```

```
# 52. What are keywords in Python?
Reserved words with special meaning.

Example: if, else, for
```

```
# 53. How can you see all Python keywords?
import keyword
print(keyword.kwlist)
```

```
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else
```

```
# 54. What is memory management in Python?
Python automatically allocates and frees memory.
```

```
# 55. What is garbage collection?
delete unused objects
a = 10
a = None  # memory freed
```

```
# 56. What is indentation in Python?
Spaces that define code blocks.
```

```
# 57. Why is indentation important?
Without indentation, Python raises error.
```

```
# 58. What is a comment?
Text ignored by interpreter.
ex ->
# This is a comment
```

```
# 59. Difference between single-line and multi-line comments.
# Single-line
"""
Multi-line
"""
```

```
# 60. What is an expression?
produces a value
5 + 3
```

```
# 61. What is a statement?
performs an action
x = 10
```

```
# 62. Difference between expression and statement.
Expression → returns value

Statement → performs action
```

```
# 63. What are literals in Python?
fixed values
10, 3.5, "Hello"
```

```
# 64. What are data types?
in which types of data stored
```

```
# 65. How many built-in data types are there?
int
float
str
list
```

```
tuple
set
dict
bool
```

```
# 66. What is type() function?
returns data type
type(10)
```

```
# 67. What is type casting?
converting one types to another
int("5")
```

```
# 68. Difference between implicit and explicit type casting.
| Implicit    | Explicit  |
| ----------- | --------- |
| Automatic   | Manual    |
| int → float | str → int |
```

```
# 69. What is input() function?
takes user input
name = input("Enter your name: ")
age = int(input("Enter your age: "))
```

```
# 70. What does input() return by default?
always return string
```

```
# 71. Difference between print() and input().
print() → displays output
input() → takes input from user
```

```
# 72. What are operators?
symbols performing operators
```

```
# 73. Types of operators in Python.
Arithmetic
Relational
Logical
Bitwise
Assignment
membership
```

```
# 74. Difference between = and ==.
= assigns
== compares
```

```
# 75. Difference between is and ==.
is checks memory
== checks value
```

```
# 76. What are relational operators?
> < >= <= == !=
```

```
# 77. What are logical operators?
and, or, not
```

```
# 78. What is short-circuiting in logical operators?
stops evaluation early.
False and print("Hi")  # Not printed
True or print("Hi")  # Printed
```

```
# 79. What are bitwise operators?
& | ^ << >>
```

```
# 80. What is a conditional statement?
Used for decision making.
```

```
# 81. What is if statement?
Executes when condition is true.
```

```
# 82. What is elif?
Checks another condition.
```

```
# 83. What is else?
Runs if all conditions fail.
```

```
# 84. Can if exist without else?
yes, it is possible
```

```
# 85. What is nested if?
if inside if
```

```
# 86. What is a loop?
repeats code.
```

```
# 87. Why are loops used?
avoid repetition
```

```
# 88. What is a for loop?
used for sequences
```

```
# 89. What is a while loop?
it runs until condition is true
```

```
# 90. Difference between for and while loop.
| for              | while   |
| ---------------- | ------- |
| Known iterations | Unknown |
```

```
# 91. What is infinite loop?
never ends
```

```
# 92. What is break statement?
stops loop completely
```

```
# 93. What is continue statement?
it skips iteration
```

```
# 94. What is pass statement?
placeholder
```

```
# 95. What is range() function?
generates numbers
```

```
# 96. What does range(1, 10, 2) produce?
it generates 1, 3, 5, 7, 9
```

```
# 97. Can range() be used in while loop?
yes, it is possible
```

```
# 98. What is loop else?
executes if loop ends normally
```

```
# 99. When does loop else execute?
when no break occurs
```

```
# 100. What is a function?
reusable block of code
```

```
# 101. Why do we use functions?
code reuse, readability
```

```
# 102. How do you define a function?
using def keyword
```

```
# 103. What is the syntax of a function?
def function_name():
    pass
```

```
# 104. What is function calling?
it is a executing function
```

```
# 105. Difference between defining and calling a function.
create vs execute
```

```
# 106. What are parameters?
variables in function definition
```

```
# 107. What are arguments?
value passed to function
```

```
# 108. Difference between parameters and arguments.
definition vs Call
```

```
# 109. What is a return statement?
sends value back
```

```
# 110. Can a function return multiple values?
returned as tuple
```

```
# 111. What happens if a function has no return?
it return none.
```

```
# 112. What is default parameter?
it is a predefined value
```

```
# 113. How do default arguments work?
it is used if argument not passed
```

```
# 114. What are positional arguments?
it is based on the position
```

```
# 115. What are keyword arguments?
it is based on name
```

```
# 116. What is *args?
*args is a variable positional arguments
```

```
# 117. Why is *args used?
*args is used to pass multiple arguments and flexibility
```

```
# 118. What is **kwargs?
it is a variable keyword arguments
```

```
# 119. Why is **kwargs used?
it is used to pass dictionary-style data
```

```
# 120. Can we use both *args and **kwargs together?
yes, we can use both of them
```

```
# 121. What is variable scope?
it is a area where variable is accessible
```

```
# 122. What is local scope?
inside function
```

```
# 123. What is global scope?
outside function
```

```
# 124. Difference between local and global variables.
function vs entire program
```

```
# 125. What is the global keyword?
it is used to modify global variable
```

```
# 126. Why do we need global keyword?
we need it to change global value
```

```
# 127. Can a function access a global variable?
yes, it is possible
```

```
# 128. Can a function modify a global variable?
yes , using global
```

```
# 129. What is nested function?
function inside function
```

```
# 130. Can we define a function inside another function?
yes, we can define function inside another function
```

```
# 131. What is function recursion?
function calling itself
```

```
# 132. What is base condition in recursion?
it stops recursion
```

```
# 133. What happens if base condition is missing?
infinite recursion
```

```
# 134. What is stack overflow?
it is memory exhausted
```

```
# 135. What is lambda function?
anonymous one line function
```

```
# 136. Difference between lambda and normal function.
| Lambda   | Normal         |
| -------- | -------------- |
| One line | Multiple lines |
```

```
# 137. Can a function be assigned to a variable?
yes, we can assign function to variable
```

```
# 138. What is a first-class function?
a function treated as variables
```

```
# 139. What is a higher-order function?
takes or returns function
```