---

1. Function with No Arguments and No Return

```
def greet():
    print("Hello, welcome to Python functions!")

greet()
```

---

2. Function with Positional Arguments

```
def add(a, b):
    result = a + b
    print("Sum:", result)

add(5, 10)
```

---

3. Function with Default Arguments

```
def greet(name="Guest"):
    print("Hello", name)

greet("Alice")
greet()
```

---

4. Function with Keyword Arguments

```
def student_info(name, age):
    print("Name:", name)
    print("Age:", age)

student_info(age=20, name="Ravi")
```

---

5. Function with Variable-Length Arguments (`*args`)

```
def total_sum(*numbers):
    result = sum(numbers)
    print("Total Sum:", result)

total_sum(1, 2, 3)
total_sum(10, 20, 30, 40)
```

---

6. Function Returning a Value

```
def square(num):
    return num * num

result = square(4)
print("Square:", result)
```

---

7. Function with Multiple Return Values

```
def calculate(a, b):
    return a + b, a - b, a * b

add, sub, mul = calculate(10, 5)
print("Add:", add)
print("Subtract:", sub)
print("Multiply:", mul)
```

---

8. Function with Default and Positional Arguments

```
def greet(name, message="Welcome!"):
    print(name + ", " + message)

greet("John")
greet("Alice", "Good to see you!")
```

---

9. Function with `*args` (Variable-Length Positional Arguments)

```
def multiply(*nums):
    result = 1
    for n in nums:
        result *= n
    print("Product:", result)

multiply(2, 3, 4)
```

---

10. Function with `**kwargs` (Variable-Length Keyword Arguments)

```
def print_info(**kwargs):
    for key, value in kwargs.items():
        print(f"{key}: {value}")

print_info(name="Anu", age=25, city="Delhi")
```

---

11. Lambda Function Example

```
square = lambda x: x * x
print("Square:", square(7))
```

---

12. Function Calling Another Function

```
def message():
    return "Hello"

def greet():
    print(message(), "from another function!")

greet()
```

---

13. Recursive Function (Factorial)

```
def factorial(n):
    if n == 0:
        return 1
    return n * factorial(n - 1)

print("Factorial of 5:", factorial(5))
```

---

14. Function with a List Argument

```
def print_list(items):
    for item in items:
        print(item)

print_list(["apple", "banana", "cherry"])
```

---

15. Function with Conditional Logic

```
def is_even(num):
    if num % 2 == 0:
        return True
    else:
        return False

print("Is 10 even?", is_even(10))
print("Is 7 even?", is_even(7))
```

---

output

Program 1 Output:

```
Hello, welcome to Python functions!
```

Program 2 Output:

```
Sum: 15
```

Program 3 Output:

```
Hello Alice
Hello Guest
```

Program 4 Output:

```
Name: Ravi
Age: 20
```

Program 5 Output:

```
Total Sum: 6
Total Sum: 100
```

Program 6 Output:

```
Square: 16
```

Program 7 Output:

```
Add: 15
Subtract: 5
Multiply: 50
```

Program 8 Output:

```
John, Welcome!
Alice, Good to see you!
```

Program 9 Output:

```
Product: 24
```

Program 10 Output:

```
name: Anu
age: 25
city: Delhi
```

Program 11 Output:

```
Square: 49
```

Program 12 Output:

```
Hello from another function!
```

Program 13 Output:

```
Factorial of 5: 120
```

Program 14 Output:

```
apple
banana
cherry
```

Program 15 Output:

```
Is 10 even? True
Is 7 even? False
```

---

recursive functions(quiz)

---

1. What is recursion?

A) Repeating a function with different values
B) A function calling another function
C) A function calling itself
D) Looping inside a function

---

2. What is the base case in recursion?

A) The condition to end the loop
B) The smallest input that ends recursion
C) A loop inside recursion
D) A function that doesn't return

---

3. What happens if a recursive function doesn’t have a base case?

A) It runs once
B) It stops automatically
C) It runs forever or crashes
D) It returns 0

---

4. What is the output of this code?

```
def fun(n):
```

```
    if n == 0:
        return 0
    else:
        return n + fun(n-1)

print(fun(3))
```

A) 3
B) 6
C) 0
D) Error

---

5. Recursive function must:

A) Use loops
B) Have a return statement
C) Call itself
D) Both B and C

---

6. Which of these is a classic recursive problem?

A) Multiplication
B) Sorting
C) Factorial
D) Input taking

---

7. What is the output of this code?

```
def fact(n):
    if n == 1:
        return 1
    return n * fact(n-1)

print(fact(4))
```

A) 4
B) 10
C) 24
D) 0

---

8. Which keyword is used to exit recursion?

A) break
B) exit
C) return
D) stop

---

9. What does this function compute?

```
def sum(n):
    if n == 0:
        return 0
    return n + sum(n - 1)
```

A) Square of n
B) Sum from 1 to n
C) Fibonacci number
D) Factorial of n

---

10. What is the output of this code?

```
def hello(n):
    if n == 0:
        return
    print("Hi")
    hello(n-1)

hello(3)
```

A) Hi
B) Hi Hi
C) Hi Hi Hi
D) Error

---

11. Recursion is similar to which looping structure?

A) for loop
B) while loop
C) do-while loop
D) All of the above

---

12. What is the depth of recursion?

A) Number of loops inside a function
B) Number of variables used
C) Number of times the function calls itself
D) The indentation level

---

13. Recursion is best suited for:

A) Problems with repetitive steps
B) Problems with unknown inputs
C) Problems with multiple functions
D) Problems that require randomization

---

14. What will this return?

```
def power(a, b):
    if b == 0:
        return 1
    return a * power(a, b - 1)

print(power(2, 3))
```

A) 6
B) 8
C) 9
D) 4

---

15. Recursive functions use which data structure?

A) Queue
B) Array
C) Stack
D) List

---

16. Recursive Fibonacci function can be slow because:

A) It has too many loops
B) It repeats calculations
C) It doesn’t return values
D) Python doesn't support it

---

17. Which of the following is true about recursion in Python?

A) Only works with integers
B) Must use return
C) Requires a base case
D) Both B and C

---

18. What is tail recursion?

A) Function calls itself at the start
B) Recursive call is the last operation in function
C) Function ends with return

D) None of the above

---

19. How many times will this run?

```
def fun(n):
    if n == 0:
        return
    fun(n-1)

fun(5)
```

A) 4
B) 5
C) 6
D) Infinite

---

20. What is the output?

```
def rev_print(n):
    if n == 0:
        return
    print(n)
    rev_print(n-1)

rev_print(3)
```

A) 3 2 1
B) 1 2 3
C) 3 2 1 0
D) 0 1 2 3

21.def reverse_string(s):

```
  if len(s) == 0:

     return s

  return reverse_string(s[1:]) + s[0]


print("Reversed string:", reverse_string("hello"))
```