PYTHON

FUNCTIONS – HARD(LEVEL)

1. What will the following print?

```
def f(a=[]):
  a.append(1)
  return a
print(f())
print(f())
```

A) [1] [1]
B) [1] [1,1]
C) [1,1] [1,1]
D) Error
Answer: B

2. Why does the above happen?
A) Default arguments evaluated once at definition
B) Default arguments re-evaluated every call
C) Python ignores mutable defaults
D) Lists cannot be default arguments
Answer: A

3. Which of the following avoids the mutable default argument issue?
A) Use None and assign inside function
B) Always use integers
C) Use global variables
D) None
Answer: A

4. What is true about default arguments?
A) Can be mutable objects
B) Can be immutable
C) Evaluated at function call
D) A & B, but evaluated at definition
Answer: D

5. Which of these will print [1] [1]?

```
def f(a=None):
  if a is None:
    a = []
```

```
    a.append(1)
  return a
```

A) Correct
B) Incorrect
Answer: A

6. Which statement is true?
   A) Default arguments are safe for immutable types
   B) Default arguments always safe for mutable types
   C) Python prohibits mutable defaults
   D) Default arguments must be numbers
   Answer: A

7. What is the output?

```
def f(x, l=[]):
  l.append(x)
  return l
print(f(1))
print(f(2,[]))
print(f(3))
```

   A) [1] [2] [1,3]
   B) [1] [2] [1,2,3]
   C) [1] [2] [3]
   D) [1] [2] [3]
   Answer: A

8. Mutable default arguments are evaluated:
   A) Every call
   B) Only at function definition
   C) Never
   D) Every time you import module
   Answer: B

9. What is the risk of mutable default arguments?
   A) Unintended shared state between calls
   B) Memory leak
   C) Syntax error
   D) Performance issue
   Answer: A

10. Which is safe way to define default empty dictionary?
    A) def f(d={}): ...
    B) def f(d=None): d={} if d is None ...
    C) def f(d=list()): ...

D) def f(d=dic()): ...
Answer: B

11. Which of these will cause RecursionError?
    A) Recursive function with no base case
    B) Function with correct base case
    C) Iterative function
    D) Tail recursion
    Answer: A

12. Python **does not optimize tail recursion**, because:
    A) Design decision; no TCO
    B) Memory issue
    C) Cannot implement
    D) It does optimize
    Answer: A

13. Which of the following is correct factorial using recursion?

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

    A) Correct
    B) Incorrect
    Answer: A

14. Recursive Fibonacci function is slow due to:
    A) Multiple repeated calls
    B) Python inefficiency
    C) Memory allocation
    D) Function annotation
    Answer: A

15. Which technique optimizes recursive functions?
    A) Memoization
    B) Tail recursion
    C) Iteration
    D) Both A & C
    Answer: D

16. What is true about recursion in Python?
    A) Each call consumes stack space
    B) Stack overflow possible
    C) Recursive functions can have default arguments
    D) All of the above
    Answer: D

17. Which of the following is **tail-recursive**?

```
def f(n, acc=1):
```

```
if n==0: return acc
return f(n-1, acc*n)
```

A) Tail-recursive
B) Not tail-recursive
Answer: A

18. What happens if recursion depth exceeds limit?
A) RecursionError
B) Program crashes silently
C) Python converts to loop
D) None
Answer: A

19. Which statement is true about recursive functions returning functions?
A) Can return closures
B) Cannot return anything
C) Only returns numbers
D) Only works with lambda
Answer: A

20. What is a potential issue with deeply recursive functions in Python?
A) Stack overflow / RecursionError
B) Memory leak
C) Performance slowdown
D) All of the above
Answer: D

21. Which is true about closures?
A) Retain outer variables after function returns
B) Forget outer variables
C) Only work with globals
D) Only for lambda
Answer: A

22. Which keyword allows modifying outer function variable?
A) nonlocal
B) global
C) local
D) static
Answer: A

23. What is printed?

```
def outer(x):
  def inner(y):
    return x+y
  return inner
f = outer(3)
```

```
print(f(4))
```

A) 7
B) 34
C) Error
D) None
Answer: A

24. Nested function can access:
A) Outer function local variables
B) Global variables
C) Both
D) Only its own locals
Answer: C

25. Which is true?

```
def f(x):
  def g(y):
    return x+y
 return g
```

A) g is a closure
B) g is global
C) g loses x
D) g is lambda
Answer: A

26. Which of the following is **not a closure**?
A) Nested function not returned
B) Nested function returned accessing outer variables
C) Lambda capturing outer variable
D) Returned function using outer variables
Answer: A

27. Closures can be used to:
A) Maintain state
B) Implement decorators
C) Both
D) None
Answer: C

28. Can closures modify mutable outer variables without nonlocal?
A) Yes, for mutable objects
B) No
Answer: A

29. What is true about closures and garbage collection?
A) Retained until all references lost
B) Immediately deleted

C) Cannot retain mutable objects
D) None
Answer: A

30. Which is true about nested functions?
A) Can return functions
B) Can be closures
C) Can access global variables
D) All of the above
Answer: D

31. What is a decorator?
A) Function modifying another function
B) Class
C) Lambda
D) Variable
Answer: A

32. Which syntax applies a decorator?
A) @decorator above function
B) func = decorator(func)
C) Both
D) None
Answer: C

33. Decorators can:
A) Modify input/output
B) Log function calls
C) Measure execution time
D) All of the above
Answer: D

34. Which of these applies multiple decorators?
A) Decorators stacked above function; bottom-up execution
B) Only one decorator allowed
C) Cannot be used on lambda
D) Must apply manually inside function
Answer: A

35. Decorators with arguments require:
A) Extra wrapper function
B) Global variable
C) Default arguments
D) Lambda only
Answer: A

36. Higher-order function:
A) Takes function as argument or returns function
B) Returns only numbers
C) Works with lambda only

D) Global function
Answer: A

37. Which of the following is true?

```
def decorator(f):
    def wrapper(*args, **kwargs):
        print("Before")
        result = f(*args, **kwargs)
    print("After")
    return result
 return wrapper
```

A) Prints messages before and after function call
B) Prints only after
C) Does not print
D) Error
Answer: A

38. Can decorators return functions with modified signatures?
A) Yes
B) No
Answer: A

39. Which is true about functools.wraps?
A) Preserves original function metadata
B) Modifies function
C) Not needed
D) Only works with lambda
Answer: A

40. Decorators can be applied to:
A) Regular functions
B) Lambda functions
C) Methods
D) All of the above
Answer: D

41. Which of these is a correct lambda?
A) lambda x: x*2
B) lambda x {x*2}
C) lambda x: return x*2
D) lambda(x): x*2
Answer: A

42. Lambda functions can:
A) Only have one expression
B) Multiple statements

C) Cannot return
D) Only global variables
Answer: A

43. Lambda can be returned from function?
A) Yes
B) No
Answer: A

44. Function annotations can:
A) Provide type hints
B) Modify behavior
C) Return values
D) Replace arguments
Answer: A

45. Which is true about introspection?
A) Functions are objects
B) Functions have __name__
C) Functions have __doc__
D) All of the above
Answer: D

46. Which statement is true about generators?
A) Use yield
B) Maintain state
C) Can be iterated only once
D) All of the above
Answer: D

47. What does next(gen) do?
A) Gets next value from generator
B) Resets generator
C) Converts to list
D) Deletes generator
Answer: A

48. Which of these is true about generator functions?
A) Can be closures
B) Can accept arguments
C) Can be decorated
D) All of the above
Answer: D

49. Which is true about Python functions as first-class objects?
A) Can be assigned to variables
B) Can be passed as arguments
C) Can be returned from other functions
D) All of the above
Answer: D

50. Which statement is true about function scopes?
    A) LEGB: Local → Enclosing → Global → Built-in
    B) Python evaluates variables bottom-up
    C) Functions cannot access enclosing scope
    D) None
    Answer: A