PYTHON TEST - 3.4 (FLOW OF EXECUTION)

Total points 50/50

Flow of Execution in a Function Call

STUDENT NAME *

VIVA

✓ 1. In Python, the flow of execution always starts from: * 1/1

- a) First function definition
- b) First function call
- [✓ c) First line of the program] ✓
- d) Last line of the program

✓ 2. The flow of execution enters a function when: * 1/1

- a) The function is defined
- [✓ b) The function is called] ✓
- c) The program ends
- d) None of these

✓ 3. When a function is called, control of the program: * 1/1

- [✓ a) Ends immediately] ✓
- b) Passes to the function body
- c) Passes to main() directly
- d) Skips the function

✓ 4. After completing execution of a function, the control: * 1/1

- [✓ a) Returns to the caller] ✓
- b) Stays inside the function
- c) Ends the program
- d) Goes to global scope permanently

✓ 5. A function definition itself does not run until: * 1/1

- a) It is imported
- [✓ b) It is called] ✓
- c) Program starts
- d) It is compiled

✓ 6. The flow of execution in Python is generally: * 1/1

- a) Random
- [✓ b) Sequential (Top to Bottom)] ✓
- c) Reverse Order
- d) Depends on compiler

✓ 7. If a function is defined after its call, Python will: * 1/1

- a) Run normally
- [✓ b) Show an error] ✓
- c) Skip the function
- d) Execute partially

✓ 8. A function may be executed multiple times depending on: * 1/1

- ○ a) How many times it is defined
- ● b) How many times it is called ✓
- ○ c) Both a and b
- ○ d) None of these

✓ 9. The execution inside a function stops when: * 1/1

- ○ a) It reaches the return statement
- ○ b) The last line of the function executes
- ○ c) An exception occurs
- ● d) All of the above ✓

✓ 10. When a function is called within another function, execution: * 1/1

- ○ a) Pauses the outer function
- ● b) Completes the inner function first ✓
- ○ c) Immediately ends the program
- ○ d) Skips the inner function

✓ 11. Python uses which mechanism for managing function calls? * 1/1

[✓ a) Stack] ✓

b) Queue

c) Linked list

d) Array

✓ 12. Each function call in Python creates: * 1/1

a) A new process

b) A new thread

[✓ c) A new stack frame] ✓

d) A new object

✓ 13. After the function execution completes, its stack frame is: * 1/1

[✓ a) Destroyed] ✓

b) Sent to caller

c) Kept permanently

d) Saved in memory forever

✓ 14. Recursive functions use: * 1/1

[✓ a) Multiple stack frames] ✓

b) Single stack frame only

c) Infinite memory

d) No memory

✓ 15. If recursion goes too deep, Python raises: * 1/1

a) MemoryError

[✓ b) RuntimeError (RecursionError)] ✓

c) SyntaxError

d) NameError

✓ 16. When no return statement is given, a function returns: * 1/1

a) 0

[✓ b) None] ✓

c) Error

d) Empty string

✓ 17. A return statement in a function: * 1/1

- a) Ends the function immediately
- b) Transfers control back to caller
- c) Returns a value (if provided)
- [✓ d) All of the above] ✓

✓ 18. Execution continues after function call at: * 1/1

- a) Beginning of program
- [✓ b) Line after the function call] ✓
- c) End of program
- d) Inside the function again

✓ 19. If multiple return statements are present, Python executes: * 1/1

- a) All of them sequentially
- [✓ b) The first one encountered] ✓
- c) Only the last one
- d) None of these

✓ 20. A function without return but with print: * 1/1

- [✓ a) Returns None] ✓
- b) Returns printed value
- c) Returns 0
- d) Returns error

✓ 21. What is the output? * 1/1

```
def greet():
    print("Hello")
greet()
print("Bye")
```

- [✓ a) Hello Bye] ✓
- b) Bye Hello
- c) Error
- d) Nothing

✓ 22. Flow of execution in nested function calls follows: * 1/1

- [✓ a) LIFO (Last In First Out)] ✓
- b) FIFO (First In First Out)
- c) Random Order
- d) None

✓ 23. What will be printed? * 1/1

```
def f1():

    print("f1")

    f2()

def f2():

    print("f2")

f1()
```

- ○ a) f2 f1
- [✓ b) f1 f2] ✓
- ○ c) Error
- ○ d) None

✓ 24. Execution always resumes after a function call at: * 1/1

- [✓ a) Next statement in caller function] ✓
- ○ b) Start of main program
- ○ c) Start of function definition
- ○ d) End of file

✓ 25. If a function returns another function, flow of execution: * 1/1

- a) Ends
- [✓ b) Passes function object back to caller] ✓
- c) Skips it
- d) Shows error

✓ 26. In recursion, flow of execution: * 1/1

- a) Goes infinitely
- [✓ b) Returns after base condition is met] ✓
- c) Never returns
- d) Crashes always

✓ 27. Flow of execution in indirect recursion stops when: * 1/1

- a) Stack overflow occurs
- [✓ b) Base condition is met] ✓
- c) Program ends
- d) Syntax error occurs

✓ 28. Flow of execution in indirect recursion stops when: * 1/1

- a) Stack overflow occurs
- [✓ b) Base condition is met] ✓
- c) Program ends
- d) Syntax error occurs

✓ 29. If recursion has no base case, execution ends with: * 1/1

- a) NameError
- [✓ b) RecursionError] ✓
- c) SyntaxError
- d) ValueError

✓ 30. When multiple functions are called sequentially, execution: * 1/1

- a) Jumps randomly
- [✓ b) Completes one by one in order] ✓
- c) Runs all at once
- d) None of these

✓ 31. Python maintains function calls in: * 1/1

- a) Heap memory
- [● b) Call stack] ✓
- c) Queue
- d) File system

✓ 32. Which executes first? * 1/1

- a) Caller function
- [● b) Called function] ✓
- c) Both together
- o d) None

✓ 33. When execution returns from a function, local variables: * 1/1

- [● a) Are destroyed] ✓
- b) Remain in memory forever
- c) Move to global scope
- d) Become constants

✓ 34. Functions are executed in: * 1/1

- ○ a) Compile time
- [✓ ● b) Runtime] ✓
- ○ c) Both a and b
- ○ d) None

✓ 35. Execution of a Python function is suspended using: * 1/1

- [✓ ● a) yield keyword] ✓
- ○ b) return keyword
- ○ c) break keyword
- ○ d) pass keyword

✓ 36. Flow of execution inside a function starts at first line of function. * 1/1

- [✓ ● True] ✓
- ○ False

✓ 37.Functions can change execution order of a program. * 1/1

- [✓ ● True] ✓
- ○ False

✓ 38. If function is not called, it still executes. * 1/1

- True
- [✓ False] ✓

✓ 39. A return statement may appear anywhere in a function. * 1/1

- [✓ True] ✓
- False

✓ 40. The execution of a program always ends inside the last function call. * 1/1

- True
- [✓ False] ✓

✓ 41. What will happen? * 1/1

```
def hello():

  print("Hi")

x = hello

x()
```

- a) Error
- [✓ b) Hi] ✓
- c) None
- d) Nothing

✓ 42. Execution order in Python follows: * 1/1

[✓ a) Top-down] ✓

b) Bottom-up

c) Parallel

d) Random

✓ 43. If a function has multiple return values separated by commas, Python returns: * 1/1

a) List

[✓ b) Tuple] ✓

c) Dictionary

d) Error

✓ 44. After function call ends, Python interpreter: * 1/1

[✓ a) Immediately executes next statement] ✓

b) Stops the program

c) Waits for user input

d) None

✓ 45. Flow of execution can be visualized using: * 1/1

○ a) Call stack
○ b) Flowchart
○ c) Debugger
[✓ d) All of these] ✓

✓ 46. Each function call creates its own local scope. * 1/1

[✓ True] ✓
○ False

✓ 47. Execution of Python function starts only when interpreter reaches the call. * 1/1

[✓ True] ✓
○ False

✓ 48. Flow of execution in Python is handled by compiler. * 1/1

○ True
[✓ False] ✓

✓ 49. Nested calls always complete inner function before outer resumes. * 1/1

[✓ True] ✓

False

✓ 50. Flow of execution in Python functions is predictable and structured. * 1/1

[✓ True] ✓

False

This content is neither created nor endorsed by Google. - Contact form owner - Terms of Service - Privacy Policy

Google Forms