**Clean Code Assessment**

**Part 1: Identify the Issue**

**Instructions: Each code snippet below violates clean code principles. Choose the best answer that identifies the issue.**

1.What is the issue with the following function?

**def c(x, y): return x+y**

a) Function is written in a single line, reducing readability.  
b) Function name is too short.  
c) Function should return a list.  
d) Function should use print instead of return.

2. What is wrong with this condition check?

**def process\_data(data):**

**if data == None:**

**return "Error"**

a) Using == None instead of is None, and missing else case.  
b) Should return a number instead of a string.  
c) Should use elif instead of if.  
d) Should use a while loop instead.

3. What issue does the following code present?

**a = 5**

**b = 10**

**c = 5**

**d = 10**

a) Repeating values instead of using constants or lists.  
b) Variables should be global.  
c) Values should be floats.  
d) Should use a dictionary instead.

4.What is the issue with this print statement?

**print("Processing");print("Completed")**

a) Using semicolons unnecessarily.  
b) print statements should be inside a function.  
c) Should use input() instead of print().  
d) Should use f-strings.

5.What is wrong with the following dictionary?

**userinfo = {**

**"nm": "John",**

**"ag": 30,**

**"cn": "USA"**

**}**

a) Non-descriptive key names.  
b) Dictionary keys should be integers.  
c) Should use a tuple instead.  
d) Should be declared inside a function.

6.What issue is present in this function?

**def calc():**

**x = 10**

**y = 20**

**z = 30**

**print(x + y + z)**

a) Hardcoded values.  
b) Should use return instead of print.  
c) Variables should be declared outside the function.  
d) Function name is unclear.

7.What problem exists in the following code?

**def process():**

**return "Processed"**

**def execute():**

**return "Executed"**

a) Functions are unrelated but placed together.  
b) Functions should use print instead of return.  
c) Functions should have parameters.  
d) Function names are too long.

8.What is the issue with this function name?

**def long\_function\_name\_that\_is\_very\_confusing\_and\_unnecessarily\_long():**

**return True**

a) Overly long function name.  
b) Should return a string instead.  
c) Should use print() instead of return.  
d) Function should take parameters.

9. What is the problem with this exception handling?

**try:**

**result = 10 / 0**

**except Exception as e:**

**print("An error occurred")**

a) Catching all exceptions without specifying the type.  
b) Should not use a try-except block.  
c) Should use finally.  
d) Should assign result outside the try block.

10. What issue does this code have?

**a = 10; b = 20; c = a + b**

a) Multiple statements on one line.  
b) Variables should be inside a function.  
c) Should use print() after the operation.  
d) Should use different variable names.

**Part 2: Match the Principle**

**Instructions: Match each principle to its correct description by writing the correct letter next to the number.**

A. Meaningful Names  
B. DRY Principle  
C. Error Handling  
D. Consistent Formatting  
E. Comments & Documentation  
F. Keeping Functions Short  
G. Using Meaningful Whitespace  
H. Refactoring Regularly  
I. Maintainability  
J. Readability

1. \_\_\_ Avoids duplicating code and encourages reusability.
2. \_\_\_ Ensures variable and function names describe their purpose clearly.
3. \_\_\_ Prevents unnecessary crashes by handling exceptions properly.
4. \_\_\_ Ensures indentation and spacing are uniform across the codebase.
5. \_\_\_ Keeps functions focused on a single task for better clarity.
6. \_\_\_ Helps developers understand code logic through proper documentation.
7. \_\_\_ Uses blank spaces and line breaks effectively for clarity.
8. \_\_\_ Improves existing code over time as requirements evolve.
9. \_\_\_ Ensures that code is easy to modify in the future.
10. \_\_\_ Helps code be easier to follow and interpret.

**Part 3: Scenario-Based Questions**

**Instructions: Read each scenario carefully and choose the best response.**

1. You find a function that is 100 lines long and performs multiple unrelated tasks. What should you do?  
   a) Leave it as is, since it works.  
   b) Break it into smaller functions, each with a single responsibility.  
   c) Just add more comments to explain each section.
2. A team member writes a variable called x1. How would you improve it?  
   a) Rename it to something more meaningful.  
   b) Add a comment explaining what x1 means.  
   c) Keep it the same since shorter names are faster to type.
3. While reviewing code, you notice repeated blocks of logic in multiple places. What is the best action?  
   a) Move the repeated logic into a separate function.  
   b) Ignore it—if it works, don’t fix it.  
   c) Just copy and paste the code whenever needed.
4. Which best describes the DRY principle?  
   a) Write every piece of knowledge only once.  
   b) Add more comments to describe repeated code.  
   c) Repeat important code to make it more visible.
5. If a project has inconsistent formatting, what is the best way to fix it?  
   a) Use automated tools like linters to enforce consistency.  
   b) Ask every developer to manually fix their code.  
   c) Ignore it since formatting doesn’t affect execution.
6. Why should we refactor our code regularly?  
   a) To make sure the code never needs updating.  
   b) To improve readability, maintainability, and efficiency.  
   c) To add unnecessary complexity to our project.