Q1: What is the difference between while loop and for loop?

While Loop:
- Used when the number of iterations is unknown beforehand
- Continues execution as long as a condition remains True
- Requires explicit initialization and increment/decrement
- Risk of infinite loops if condition never becomes False

```
#example
count = 0
while count < 5:
    print(f"Count: {count}")
    count += 1
```

For Loop:
- Used when iterating over a sequence (list, tuple, string, range, etc.)
- Known number of iterations
- Automatically handles iteration through sequence
- More Pythonic for sequence traversal

```
#examples
for i in range(5):  # Iterate 5 times
    print(f"Number: {i}")
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:  # Iterate through list
    print(fruit)
```

Q2: What is the difference between loop-else structure and if-else structure?

Loop-else:
- The `else` clause executes ONLY if the loop completes normally (no `break`)
- Useful for search operations - execute code when item not found
- Not commonly used but can make code more readable

```
#example
numbers = [1, 3, 5, 7, 9]
for num in numbers:
    if num % 2 == 0:
        print(f"Found even number: {num}")
        break
else:
    print("No even numbers found")  # Executes if no break occurred
```

If-else:**
- Standard conditional branching
- Executes based on Boolean condition evaluation
- Fundamental control structure

```
# example
age = 18
if age >= 18:
    print("You can vote")
else:
    print("You cannot vote")
```

Q3: What is the significance of if-elif-else structure over if-else structure?

if-elif-else advantages:
- More efficient - stops checking once a True condition is found
- Cleaner and more readable for multiple conditions
- Avoids unnecessary condition checks

```
# Correct: if-elif-else
grade = 85
if grade >= 90:
    print("A")
elif grade >= 80:
    print("B")  # Stops here, doesn't check remaining conditions
elif grade >= 70:
    print("C")
else:
    print("F")

# Bad: Multiple if-else (less efficient)
if grade >= 90:
    print("A")
if grade >= 80 and grade < 90:  # Always checks this even if first was True
    print("B")
if grade >= 70 and grade < 80:
    print("C")
```

Q4: What is the difference between break and continue statements?

Break:
- Immediately terminates the entire loop
- Control moves to the first statement after the loop
- Useful for early exit when condition met

```
# Break example
for i in range(10):
    if i == 5:
        break  # Loop stops completely when i reaches 5
    print(i)
# Output: 0 1 2 3 4
```

Continue:
- Skips the current iteration and moves to the next one
- Loop continues with next iteration
- Useful for skipping specific cases

```
# Continue example
for i in range(5):
    if i == 2:
        continue  # Skip iteration when i is 2
    print(i)
# Output: 0 1 3 4
```

Q5: What is the significance of pass statement in functions?

Pass statement uses:
- The pass statement in Python acts as a *placeholder* for future code.
- It tells the interpreter to do nothing, allowing you to define a function, class, or loop with an empty body without causing an error.
- This is useful when you're planning the structure of your program and want to leave certain code blocks empty until you implement them later.

```
# Pass example
def future_function():
    pass
```

Q6: What is the importance of assert statement in program?

Assert statement purposes:
- Debugging aid - checks for conditions that should always be True
- Program correctness verification
- Can be disabled with `-O` flag for optimized runs

```
# Assert examples
x = 10
assert x > 0, "x should be positive"  # Runs silently if true, errors if false
```

Q7: What benefit we get if we invoke any function inside the if expression as if name==’main’.

Key benefits:
- Code executes only when script is run directly, not when imported
- Allows file to be used as both executable script and importable module
- Prevents unintended execution of test code/main logic

```
# my_module.py
def helper_function():
    print("Helper function called")
def main():
    print("This is the main program logic")
    helper_function()

if __name__ == '__main__':
    main()  # Only runs if script executed directly
# When imported: helper_function available but main() doesn't run automatically.
```

Q8: What is the difference between logical and relational operators. Comment on their output style.

Relational Operators:
- Compare values: `==`, `!=`, `<`, `>`, `<=`, `>=`
- Return Boolean values (`True`/`False`)
- Used for condition checks

```
# example
a, b = 5, 10
print(a == b)  # False
print(a < b)   # True
print(a != b)  # True
```

Logical Operators:
- Combine Boolean expressions: `and`, `or`, `not`
- Use short-circuit evaluation
- Return one of the operands, not necessarily Boolean

```
# example
x, y = 5, 10
print(x < 10 and y > 5)  # True
print(x > 10 or y > 5)   # True
print(not x == y)        # True

# Short-circuit evaluation
def expensive_function():
    print("This won't be called due to short-circuiting")
    return True

result = False and expensive_function()  # expensive_function never called
```

Q9: Output type questions on operators, scope, conditional statement(if), loops and builtin math and string functions