Part - 1

Description:

In this code snippet, we have two different ways to calculate the sum of two numbers, x and y, and store the result in the variable z. The first method uses a block structure, while the second method uses semicolons to separate the statements.

Code:

```
# Using a block structure to calculate the sum of x and y and assign it to z
z = begin
    x = 1
    y = 2
    x + y
end

# Using semicolons to calculate the sum of x and y and assign it to z
z = (x = 1; y = 2; x + y)
```

Output:

Both methods result in the variable z having the value 3, as it stores the sum of 1 and 2:
z = 3

Part - 2

Description:

This code demonstrates various Julia functions and conditional statements to compare two numbers, determine their relationship (less than, equal to, or greater than), and provides different outputs based on the inputs.

Code:

1)

```
if x < y
    println("x is less than y")
elseif x > y
    println("x is greater than y")
else
    println("x is equal to y")
end
```

2)

```
function test(x, y)
    if x < y
        println("x is less than y")
    elseif x > y
        println("x is greater than y")
    else
        println("x is equal to y")
    end
end
```

3)

```
function test(x,y)
    if x < y
        relation = "less than"
    elseif x == y
        relation = "equal to"
    else
        relation = "greater than"
    end

    println("x is ", relation, " y.")
end
```

Input:

- test(1, 2) - Comparing 1 and 2.
- test(2, 1) - Comparing 2 and 1.

Output:

- For test(1, 2), the output is "x is less than y."
- For test(2, 1), the output is "x is greater than y."

Part - 3

1. Logical AND Operations:

Code:

```
t(1) && t(2)
t(1) && f(2)
f(1) && f(2)
```

Input:

t(1) returns true
t(2) returns true
f(1) returns false
f(2) returns false

Output:

true && true evaluates to true
true && false evaluates to false
false && false evaluates to false

2. Logical OR Operations:

Code:

t(1) || t(2)
t(1) || f(2)
f(1) || f(2)

Input:

t(1) returns true
t(2) returns true
f(1) returns false
f(2) returns false

Output:

true || true evaluates to true
true || false evaluates to true
false || false evaluates to false

3. Recursive Factorial Function:

Code:

```
function fact(n::Int)
    n >= 0 || error("n must be non-negative")
    n == 0 && return 1
    n * fact(n-1)
end
```

Input:

fact(5)

Output:

fact(5) computes and returns the factorial of 5, which is 120.

4. Bitwise AND and OR Operations:

Code:

f(1) & t(2)
t(1) | t(2)

Input:

f(1) returns false
t(2) returns true

Output:

false & true evaluates to false
true | true evaluates to true

5. Conditional Assignment:

1)

Code:
true && (x = (1, 2, 3))

Input:
The condition is true.

Output:
x is assigned the tuple (1, 2, 3) because the condition is met.

2)

Code:
false && (x = (1, 2, 3))

Input:
The condition is false.

Output:

x remains unchanged, and no assignment is made.

Part - 4

1. While Loop with Global Variable and For Loop:

Code:

```
i = 1;
while i <= 3
    println(i)
    global i += 1
end

for j = 1:3
    println(j)
end
```

Output:

```
1
2
3
1
2
3
```

2. Variable 'j' Redefinition:

Code:

```
j=0
for j = 1:3
    println(j)
end
```

Output:

```
1
2
3
```

3. For Loop with Array:

Code:

```
for i in [1,4,0]
    println(i)
end
```

Output:

```
1
4
0
```

4. For Loop with Strings:

Code:

```
for s ∈ ["foo","bar","baz"]
    println(s)
end
```

Output:

```
foo
bar
baz
```

5. While Loop with `break` and For Loop:

Code:

1)

```
i = 1;
while true
    println(i)
    if i >= 3
        break
    end
    global i += 1
end
```

2)

```
for j = 1:1000
    println(j)
    if j >= 3
```

```
        break
    end
```

Output:

```
1
2
3
1
2
3
```

6. For Loop with `continue` statement:

Code:

```
for i = 1:10
    if i % 3 != 0
        continue
    end
    println(i)
end
```

Output:

```
3
6
9
```

7. Nested For Loop:

Code:

```
for i = 1:2, j = 3:4
    println((i, j))
end
```

Output:

```
(1, 3)
(1, 4)
(2, 3)
(2, 4)
```

8. Redefining `i` Inside Loop:

Code:

```
for i = 1:2, j = 3:4
    println((i, j))
    i = 0
end
```

Output:

```
(1, 3)
(1, 4)
(2, 3)
(2, 4)
```

9. Using `zip` Function in For Loop:

Code:

```
for (j, k) in zip([1 2 3], [4 5 6 7])
    println((j,k))
```

Output:

```
(1, 4)
(2, 5)
(3, 6)
```

Part - 5

1. Simple Function to Calculate Sum:

Code:

```
function f(x, y)
    x + y
end
```

Input: f(2, 3)
Output: 5

2. Assigning Function to a Variable:

Code:

```
g = f
```

Input: g(2, 3)
Output: 5

3. Function Using Unicode Symbol:

Code:

```
∑(x, y) = x + y
```

Input: ∑(2, 3)
Output: 5

4. Function Mutating an Array and Creating a New Binding:

Code:

```
function f(x, y)
    x[1] = 42
    y = 7 + y
    return y
end
```

Input: a = [4, 5, 6], b = 3; f(a, b)
Output: 10
Modified a: [42, 5, 6]
Unchanged b: 3

5. Recursive Fibonacci Function:

Code:

```
fib(n::Integer) = n ≤ 2 ? one(n) : fib(n-1) + fib(n-2)
```

Input: `fib(6)`
Output: `8`

6. Function Returning Product of `x` and `y` with Unused Code:

Code:

```
function Q(x, y)
    return x * y
    x + y
end
```

Input: Q(2, 3)
Output: 6

7. Function Returning Product of `x` and `y` with Unused Code:

Code:

```
function M(x, y)
    return x * y
    x + y
end
```

Input: M(2, 3)
Output: 6

8. Function to Calculate Hypotenuse:

Code:

```
function hypot(x, y)
    x = abs(x)
    y = abs(y)
    if x > y
        r = y / x
        return x * sqrt(1 + r * r)
    end
    if y == 0
        return zero(x)
    end
    r = x / y
    return y * sqrt(1 + r * r)
end
```

Input: hypot(3, 4)
Output: 5.0

9. Function with Explicit Return Type:

Code:

```
function S(x, y)::Int8
    return x * y
end
```

Input: typeof(S(1, 2))
Output: Int8

10. Function to Print a Variable:

Code:

```
function printx(x)
    println("x = $x")
    return nothing
end
```