Quiz Section 4: Functional Programming

Task 1 – A Barrel of Halfs

Consider the following function, which calculates half when given an **even** number but also accepts other inputs (though it doesn't perform the same behavior in those cases):

$$\text{half} : (\text{undefined} \ \cup \ \mathbb{N}) \to \mathbb{Z}$$

$$\begin{aligned}
&\text{half}(\text{undefined}) && := 0 \\
&\text{half}(n) && := n/2 && \text{if } n \text{ is even} \\
&\text{half}(n) && := -(n+1)/2 && \text{if } n \text{ is odd}
\end{aligned}$$

a) What would the declaration of this function look like in TypeScript based on the type?

b) What would the implementation of the body of this function look like in TypeScript?

Task 2 – The Rings of Pattern

Consider the following TypeScript code:

```
const maybeDouble = (t: {b: boolean, v: [boolean, bigint]}): bigint => {
    const [bool, num] = t.v;
    if (t.b) {
        if (bool) {
            return 2n * num;
        } else {
            return num;
        }
    } else {
        return 0n;
    }
};
```

How would you translate this into our math notation using pattern matching?

Task 3 – Sugar and Spice and Everything Twice

We are asked to write a function "twice" that takes a list as an argument and "returns a list of the same length but with every number in the list multiplied by 2".

a) This is an English definition of the problem, so our first step is to formalize it. Let's start by looking at examples. Fill in the blanks showing the result of applying twice to lists of different lengths.

nil	____________________
3 :: nil	____________________
2 :: 3 :: nil	____________________
1 :: 2 :: 3 :: nil	____________________

b) Now, let's write a formal definition that gives the correct output for **all** lists.

Write a formal definition of twice using recursion. The recursion can be identified by looking for a *pattern* in the answers to part (a).

c) What would the implementation of the body of this function look like in TypeScript?

d) What is a set of test inputs that would meet all of our requirements?

Task 4 – Put Your Mind To Test

a) For the following function, what is a set of test inputs that would meet all of our requirements?

```
const s = (x: bigint, y: bigint): bigint => {
  if (x >= 0n) {
    if (y >= 0n) {
        return x + y;
    } else {
        return x - y;
    }
  } else {
    return y;
  }
}
```

b) The following function allows only non-negative inputs. What is a set of test inputs that would meet all of our requirements?

```
const f = (n: bigint): bigint => {   // Note: requires n >= 0
  if (n === 0n) {
    return 0n;
  } else if (n === 1n) {
    return 1n;
  } else if (n % 2n === 1n) {  // n is > 1 and odd
    return f(n - 2n) + 1n;
  } else {                     // n is > 1 and even
    return f(n - 2n) + 3n;
  }
};
```

c) The following function claims to calculate $|x|$:

```
const abs_value = (x: bigint): bigint => {
  if (x > 1n) {
    return x;
  } else {
    return -x;
  }
};
```

Testing it on the inputs 2 and -2 would meet our requirements, but it would not identify the bug.

Which input do we need to test to see the bug? Which if our non-required (but recommended) heuristics would have found this?

Task 5 – Miami Twice

We are asked to write a function that takes a list as an argument and "returns a list of the same length but with *every other* number in the list, *starting with the first number*, multiplied by 2".

The first number in the list is at index 0, which is even; the second number in the list is at index 1, which is odd; the third number in the list is at index 2, which is even; and so on. Hence, we will call this function twice-evens because it multiples the numbers at even indexes by two and leaves those at odd indexes unchanged.

a) The definition of the problem was in English, so our first step is to formalize it. Let's start by writing this out in more detail. Fill in the blanks showing the result of applying twice-even to lists of different lengths.

nil	____________
4 :: nil	____________
3 :: 4 :: nil	____________
2 :: 3 :: 4 :: nil	____________
1 :: 2 :: 3 :: 4 :: nil	____________

b) Now, let's write a formal definition that gives the correct output for **all** lists.

Write a formal definition of twice-evens using recursion. The recursion can be identified by looking for a *pattern* in the answers to part (a). If the answer for one input does not appear related to and the one immediately before it, it could be related to an even *earlier* answer.

c) What would the implementation of the body of this function look like in TypeScript?

d) What is a set of test inputs that would meet all of our requirements?