1. HASK1: Simple Functions

```haskell
largest :: String -> String -> String
largest string1 string2
    | length string1 >= length string2 = string1
    | otherwise = string2
```

2. HASK2: Recursion, Precedence, Guards

```haskell
reflect :: Integer -> Integer
reflect 0 = 0
reflect num
    | num < 0 = (-1) + reflect (num + 1)
    | num > 0 = 1 + reflect (num - 1)
    | otherwise = 0
```

3. HASK3: Recursion, Guards

if statements:

```haskell
is_even :: Integer -> Bool
is_even num =
    if num == 0
    then True
    else is_odd (num - 1)

is_odd :: Integer -> Bool
is_odd num =
    if num == 0
    then False
    else is_even (num - 1)
```

guards:

```haskell
is_even :: Integer -> Bool
is_even num
    | num == 0 = True
    | otherwise = is_odd (num - 1)

is_odd :: Integer -> Bool
is_odd num
    | num == 0 = False
    | otherwise = is_even (num - 1)
```

pattern matching:

```haskell
is_even :: Integer -> Bool
is_even 0 = True
is_even 1 = False
is_even num = is_even (num - 2)

is_odd :: Integer -> Bool
is_odd 0 = False
is_odd 1 = True
is_odd num = is_odd (num - 2)
```

4. HASK4: Where, Tuples

```haskell
quad :: Double -> Double -> Double -> (Double, Double)
quad a b c
    | a == 0 = (0, 0)
    | dis >= 0 = (posRoot, negRoot)
    | otherwise = (0, 0)
    where
        dis = b * b - 4 * a * c
        disSqrt = sqrt dis
        posRoot = (-b + disSqrt) / (2 * a)
        negRoot = (-b - disSqrt) / (2 * a)
```

5. HASK5: Recursion, Where, Helper Functions

```haskell
sum_is_divisible :: Integer -> Integer -> Integer -> Bool
sum_is_divisible a b c
    | ((sum_range a b) `mod` c == 0) = True
    | otherwise = False
    where
        sum_range :: Integer -> Integer -> Integer
        sum_range x y
            | x > y = 0
            | otherwise = x + sum_range (x + 1) y
```

6. HASK6: Simple List Processing, Recursion

```haskell
find_min lst
    | lst == [] = error "empty list"
    | length lst == 1 = first
    | otherwise = min first (find_min rest)
    where
        first = head lst
        rest = tail lst
```

7. HASK7: List Comprehensions
    a.

```haskell
all_factors :: Integer -> [Integer]
all_factors n = [x | x <- [1..n], n `mod` x == 0]
```

    b.

```haskell
perfect_numbers :: [Integer]
perfect_numbers = [n | n <- [1..], sum (init (all_factors n)) == n]
```

8. HASK8: Recursion, Pattern Matching

```haskell
count_occurrences :: Eq a => [a] -> [a] -> Integer
count_occurrences [] _ = 1
count_occurrences _ [] = 0
count_occurrences (x:xs) (y:ys)
    | x == y    = count_occurrences xs ys + count_occurrences (x:xs) ys
    | otherwise = count_occurrences (x:xs) ys
```

9. HASK9: Recursion, Lists

```haskell
fibonnaci :: Int -> [Int]
fibonnaci n
    | n <= 0 = []
    | otherwise = take n fibs
    where
        fibs = 1 : 1 : zipWith (+) fibs (rest)
        rest = tail fibs
```