20CYS312 -PRINCIPLE OF PROGRAMMING LANGUAGES

Date: 06-12-2024

Name: Naren S

Roll no.: CH.EN.U4CYS22034

LAB-2

1: Functions and Types

Objective: Get familiar with basic functions.

Exercise 1: Define a function to square the given input

Code and Output Examples:

```
Naren@NAREN MINGW64 ~
$ ghci
GHCi, version 9.4.8: https://www.haskell.org/ghc/  :? for help
ghci> let square x=x*x
ghci> square 5
25
ghci> square -10

<interactive>:3:1: error:
    * No instance for (Show (Integer -> Integer))
        arising from a use of `print'
        (maybe you haven't applied a function to enough arguments?)
    * In a stmt of an interactive GHCi command: print it
ghci> square (-10)
100
ghci> square 5.5
30.25
ghci> square seven

<interactive>:6:8: error:
    Variable not in scope: seven
    Suggested fix: Perhaps use `even' (imported from Prelude)
ghci> |
```

Explaination:

A function square that takes an integer and returns its square.

Exercise 2: Define a function to find the maximum of two numbers

Code and Output Examples:

```
ghci> let maxOfTwo x y=if x>y then x else y
ghci> maxOfTwo 20 10
20
ghci> maxOfTwo 10.0 10.01
10.01
ghci> maxOfTwo 10.0 five

<interactive>:10:15: error: Variable not in scope: five
ghci> maxOfTwo 1 1
1
ghci>
```

Explaination:

A function maxoftwo that takes two integers and returns the larger one.

2.Functional Composition

Objective: Understand functions in list.

Exercise 1: Define a function doubleAndIncrement that doubles each number in a list and increments it by 1 using function composition.

Code and Output Examples:

```
ghci> let dandi = map((+1).(*2))
ghci> dandi [1,2,3]
[3,5,7]
ghci> dandi [1,2.5,3]
[3.0,6.0,7.0]
ghci> dandi [1,2.5,3,10]
[3.0,6.0,7.0,21.0]
ghci> dandi [1,2.5,3,six]

<interactive>:17:16: error:
    Variable not in scope: six
    Suggested fix: Perhaps use `sin' (imported from Prelude)
ghci>
```

Explaination:

`dandi`:

- `map` applies a function to each element in a list.
- `(*2)` doubles a number, and `(+1)` increments it by 1. These are composed with `(.)` to form `(+1).(*2)`

Exercise 2: Write a function sumOfSquares that takes a list of integers, squares each element, and returns the sum of the squares using composition.

Code and Output Examples:

```
ghci> let sumofsq = sum.map(^2)
ghci> sumofsq [1,2,3]
14
ghci> sumofsq [10,10,10]
300
ghci> sumofsq [10,10,six]

<interactive>:21:16: error:
    Variable not in scope: six
    Suggested fix: Perhaps use `sin' (imported from Prelude)
ghci>
```

Explaination:

`sumOfSquares`:

- `map (^2)` squares each number in the list.
- `sum` adds all the squared numbers together.

3. Numbers:

Objective: Learn how to write a function using recursion.

Exercise 1: Write a function factorial that calculates the factorial of a given number using recursion.

Code and Output Examples:

```
ghci> let fact 0=1; fact n=n*fact(n-1)
ghci> fact 5
120
ghci> fact 12.5
*** Exception: stack overflow
ghci> fact 3
6
ghci> fact 0
1
ghci> fact two

<interactive>:37:6: error: Variable not in scope: two
ghci>
```

Explaination:

Fact:

- `fact 0 = 1` defines the base case.
- `fact n = n * fact (n - 1)` recursively multiplies `n` with the factorial of `n-1`.

Exercise 2: Write a function power that calculates the power of a number (base raised to exponent) using recursion.

Code and Output Examples:

```
ghci> let power _ 0=1;power base exp=base^exp
ghci> power 3 2
9
ghci> power 2 10
1024
ghci> power 100000 0
1
ghci> power 2.5 3.5

<interactive>:42:1: error:
    * Could not deduce (Integral b0) arising from a use of `power'
      from the context: Fractional a
        bound by the inferred type of it :: Fractional a => a
        at <interactive>:42:1-13
      The type variable `b0' is ambiguous
      Potentially matching instances:
        instance Integral Integer -- Defined in `GHC.Real'
        instance Integral Int -- Defined in `GHC.Real'
        ...plus one other
        ...plus one instance involving out-of-scope types
        (use -fprint-potential-instances to see them all)
    * In the expression: power 2.5 3.5
      In an equation for `it': it = power 2.5 3.5
```

Explaination:

Power:

- `power _ 0 = 1`
- `power base exp = base * power base (exp - 1)` recursively multiplies `base` with the result of `power base (exp - 1)`

4) Lists:

Objective: Understand basic list operations.

Exercise 1:Write a function removeOdd that removes all odd numbers from a list.

Code and Output Examples:

```
ghci> let removeodd x=filter even x
ghci> removeodd [1,2,3,4,5,6,7]
[2,4,6]
ghci> removeodd [1,3,5,7,9]
[]
ghci> removeodd [1,2,3.5,5]

<interactive>:46:1: error:
    * Ambiguous type variable `a0' arising from a use of `print'
      prevents the constraint `(Show a0)' from being solved.
      Probable fix: use a type annotation to specify what `a0' should be.
      Potentially matching instances:
        instance Show Ordering -- Defined in `GHC.Show'
        instance Show a => Show (Maybe a) -- Defined in `GHC.Show'
        ...plus 25 others
        ...plus 14 instances involving out-of-scope types
        (use -fprint-potential-instances to see them all)
    * In a stmt of an interactive GHCi command: print it
ghci>
```

Explaination:

`removeOdd`:

- `filter even xs` filters only the even numbers from the list `xs`

Exercise 2: Write a function firstNElements that takes a number n and a list and returns the first n elements of the list.

Code and Output Examples:

```
ghci> let firstn n x=take n x
ghci> firstn 2 [1,2,3,4,5]
[1,2]
ghci> firstn 0 [1,2,3,4,5]
[]
ghci> firstn 6 [1,2,3,4,5]
[1,2,3,4,5]
ghci>
```

Explaination:

`firstNElements`:

- `take n xs` takes the first `n` elements from the list `xs`. If `n` is greater than the length of the list, it simply returns the entire list.

5) Tuples:

Objectives: Understand basic operations of tuples

Exercise 1: Define a function swap that swaps the elements of a pair (tuple with two elements)

Code and Output Examples:

```
ghci> let firstn n x=take n x
ghci> firstn 2 [1,2,3,4,5]
[1,2]
ghci> firstn 0 [1,2,3,4,5]
[]
ghci> firstn 6 [1,2,3,4,5]
[1,2,3,4,5]
ghci> let swap (x,y)=(y,x)
ghci> swap (2,3)
(3,2)
ghci> swap (213122,323.34234)
(323.34234,213122)
ghci> swap (10,0)
(0,10)
ghci> swap (10,five)

<interactive>:55:10: error: Variable not in scope: five
ghci>
```

Explaination:**`swap`**:

- The pattern `(x, y)` matches a tuple, and the function simply returns `(y, x)`

Exercise 2: Write a function addPairs that takes a list of tuples containing pairs of integers and returns a list of their sums.

Code and Output Examples:

```
ghci> let addpair x=map (\(a,b)-> a+b) x
ghci> addpair [(1,2), (3,4), (5,6)]
[3,7,11]
ghci> addpair [(1,2.5), (3,8), (5,6000)]
[3.5,11.0,6005.0]
ghci> addpair [(0,0), (0,0), (5,6)]
[0,0,11]
ghci>
```

Explaination:

`addPairs`:

- `map (\(a, b) -> a + b) xs` applies the lambda function `(\(a, b) -> a + b)` to each tuple `(a, b)` in the list `xs`, summing the two elements of each tuple.

Conclusion:

Understanding of core Haskell features, including recursion, list processing, tuple functions, and the declarative programming paradigm in this lab session.