Haskell

COMP360

Haskell Reading

- Read the Haskell tutorial at learnyouahaskell.com
- Learn Haskell up to and including Higher order functions in chapter 6

Haskell Patterns

- You can write multiple definitions of a function
- Haskell will consider them in order and execute the first on that fits
- You can put constants as the parameters in the definition. If you call the function with that value, it will execute that function definition

Fibonacci Pattern Example

- You might have written the Fibonacci function as

```
fib n = if n <= 2 then 1 else fib (n-1) + fib (n-2)
```

- It can also be written as

```
fib 1 = 1
fib 2 = 1
fib n = fib (n-1) + fib (n-2)
```

Empty List Pattern

- You can specify an empty list as a parameter. The definition will be use if it is executed with an empty list

```
mySum :: Num a => [a] -> a
mySum [] = 0
mySum str = head str + mySum (tail str )
```

Practice

- Write a Haskell method to convert a string containing a binary number to an int number

bin2dec "0101" returns 5

sumList :: [Char] -> int

Splitting Complex Values

- The parameters to a function can be specified by their parts
- A list parameter can be specified as (cat:dog)
- The ":" operator is normally used to add a new element to the beginning of a list
- Here it specifies the input list is composed of a head named cat and the tail named dog

Using Split Parameters

- Consider a function that uses the head and tail of a list
- You can describe the list as a head:tail

```
sumsqr :: Num a => [a] -> a
sumsqr [] = 0
sumsqr (cat:dog) = cat * cat + sumsqr dog
```

Values in Split Parameters

- You can put a constant in part of a split parameter to make a pattern

mytrim (' ' : cat) = mytrim cat

mytrim dog = dog

And More

- You can split an input parameter into more than two parts

myFunc (ant : bird : cat) = bird

- This returns the second element of a list
- (ant : bird : cat) can be read as a list where ant is the head of a list where bird is the head and cat the tail

Don't Care Variables

- Sometimes you are not going to use a part of the complex type, but need to specify that is there to completely describe the type
- The variable name _ or underscore represents a variable that will not be used

```
myFunc ( _ : bird : _ ) = bird
```

Practice

- Write a method that takes the union of two lists
- Note that the union should not contain duplicates

Guards

- You can put conditions on the parameters of a function

```
grade score
    | score >= 90 = "A"
    | score >= 80 = "B"
    | score >= 70 = "C"
    | otherwise   = "You flunk"
```

Calculated Guards

- The guard equations can involve more complicated equations

```
grade right total
    | 100.0 * right / total >= 90 = "A"
    | 100.0 * right / total  >= 80 = "B"
    | 100.0 * right / total >= 70 = "C"
    | otherwise   = "You flunk"
```

Simplified Guards

- Equations used in the conditions of a statement can be defined to simplify the guards

```
grade right total
    | score >= 90 = "A"
    | score >= 80 = "B"
    | score >= 70 = "C"
    | otherwise   = "You flunk"
   where score = 100.0 * right / total
```

Temporary Bindings in Functions

- You can specify variable values in an expression before the definition of a function

functionName parm = let *what = ever* in *function def*

```
border len width = let parim = 2 (len + width) in
        3 * parim
```

Haskell Reading

- Read the Haskell tutorial at learnyouahaskell.com
- Learn Haskell up to and including Higher order functions in chapter 6