Haskell

- Functions
- Types
- Scripts
- Using Haskell environment
- Guards
- **`if.. then.. else`**
- Errors
- Basic integer operations
- Naming requirements

Notes: Chapter 16,17

Sum of n integers

In C/C++/Java :

```
int sumFc (int n){
   int i,total=0;
   for(i=1; i <= n; ++i) {total=total+i;}
   return total;
}
```

In Prolog:

```
sumFc([], 0).
sumFc([FirstNumber | RestOfList], Total) :-
        sumFc(RestOfList, TotalOfRest),
        Total is FirstNumber + TotalOfRest.
```

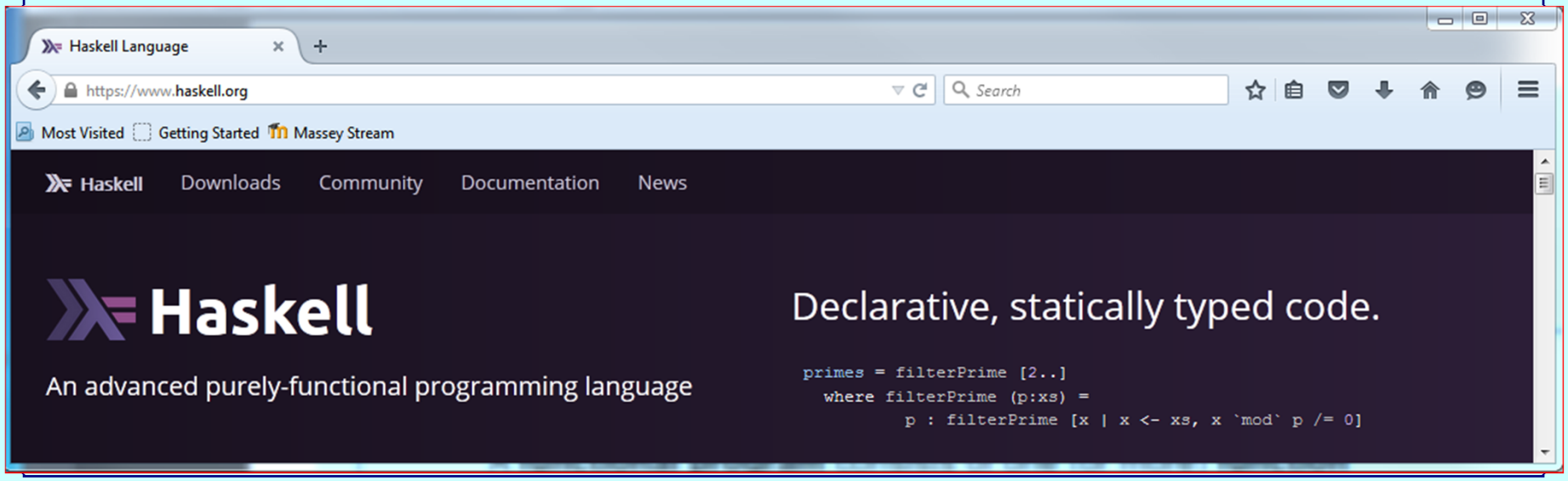
In Haskell :

```
sumFc n = sum[1..n]
```

What is Haskell?

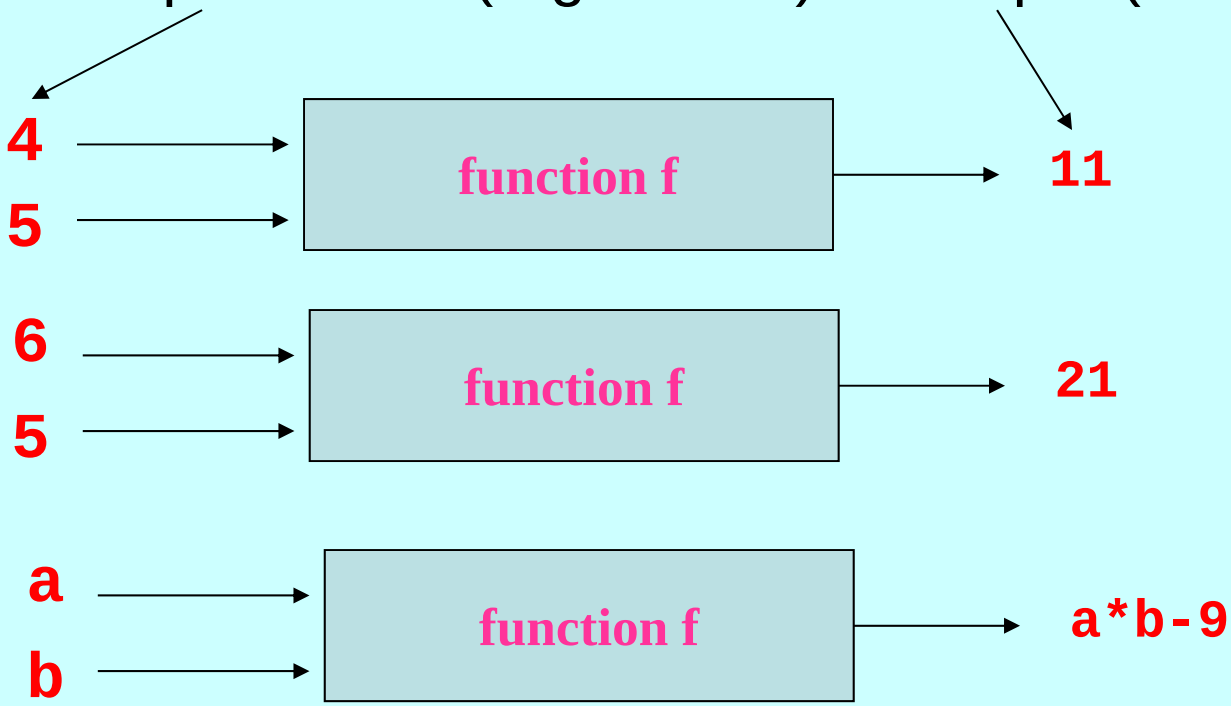
Haskell is a ***functional*** programming language.

A **functional program** consists of one (or more) **function** (expression), which is executed by evaluating the expression.



Functions

A *function* is an expression that associates to input values (arguments) an output (result).



Functions


```
functionF(a, b) = ab - 9
```

← Mathematics

```
functionF  a   b = a * b - 9
```

← **Haskell**

Quiz

Mathematics	Haskell
`f(x,y) + xy`	`f x y + x * y`
`twice(x) = 2x`	?
?	`f a + b`
$f(a,b,c) = \frac{a}{-b} + 2(-c)$	?

Script

```
-- this is a comment

{- Defines the function used

 on some previous slide -}
functionF :: Int -> Int-> Int

functionF   a    b = a * b - 9
```

Type of function

Another script

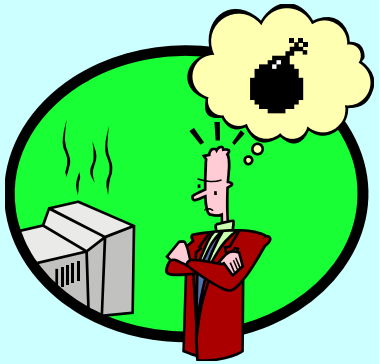
```
answer :: Int -- an integer
answer = 42


-- A function to square a  real number
square :: Float -> Float
square n = n * n

max’ :: Int -> Int -> Int
max’ n m
    | n >= m    = n
    | otherwise = m

greater :: Bool -- greater is a Boolean. Uses answer
greater = answer - 40 ≥ 6
```

Functions

In Math and functional programing

A=5, F(x)=2*x-A, F(0)is **always** -5,
F(4) is **always** 3

Other programming types—MAY NOT :



A= 5 -global variable
F(x)=2*x-A

F(0)is 2*0-5=-5

A=9

F(0)is 2*0-9=-9

So F(0) is -9 or -5 or…

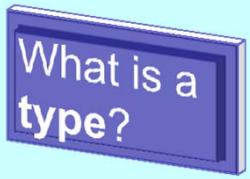
Simple types

Int -- whole numbers between -2^{31} and $2^{31}-1$

Examples: -23456, -2,0,23,1234…

Float – numbers with decimal point

Examples: -2.4, -2.0, 0.0, 1.2346



Bool – Boolean values

Examples: True, False

Char – alphanumerical symbols

Examples: 'a', 'K', '\n', '9'

Evaluating integer expressions

Expression	Value
`4 + 7 `mod` 5`	6
`4 + 32 `div` 5`	
`3 + 11 / 5`	
`2^2^3`	
`3 + 2*10 – 5`div`2`	
`3`mod`3 >= 2`div`3-5`	

Functions on integers

Function calls	Result
`sum [1..5]`	
`product [1..5]`	
`even 40`	
`odd 40`	
`lcm 4 6`	
`abs (-5)`	
`gcd 20 15`	

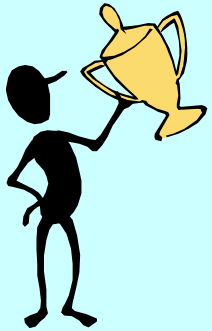
Guards-example

Consider the following game: There are 50 balls numbered from 1 to 50 in a box. Each player picks a ball at random (blindfolded). The winner is the person who gets the biggest number of points computed according to the table:

Number N	Points
under 16	N!
between 17 and 30	$500N^3$
between 31 and 50	$10^3 \ \Sigma_{i=1,\ N}\ i$

Write a Haskell script to compute the points.

Example:
Player 1: points 45 = 1035000
Player 2: points 12 = 479001600
Player 3: points 29 = 12194500



Winner: Player 2

Guards-example

Number N	Points
under 16	N!
between 17 and 30	$500N^3$
between 31 and 50	$10^3 \Sigma_{i=1, N} i$

Correct?
Good style?

--We assume **n** to be a **positive integer less 50**

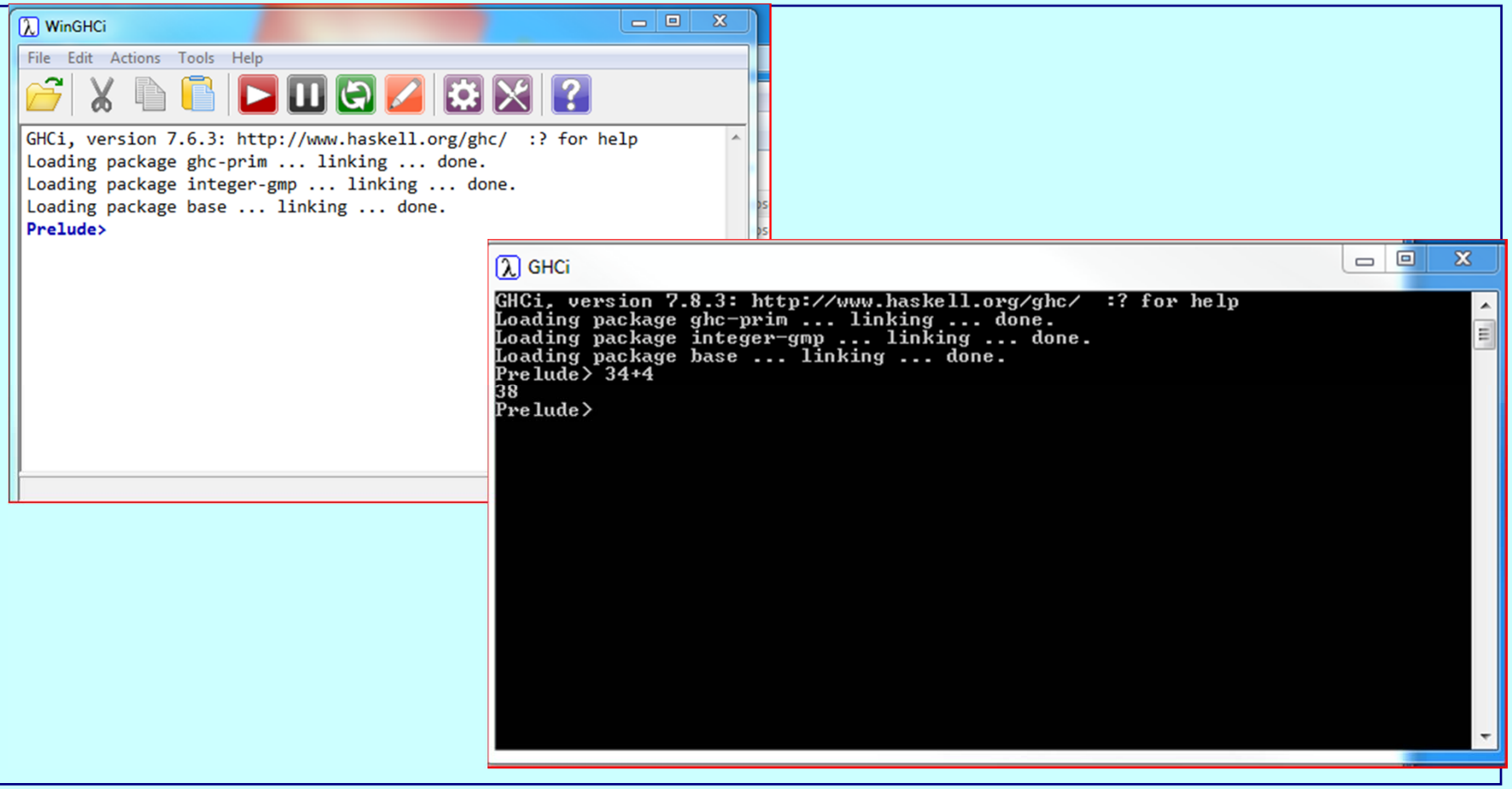
```
points n
  |n >= 31    = 1000 * sum [1..n]
  |n >= 17    = 500 * n ^ 3
  |otherwise = product [1..n]
```

```
points n
   |n  > 31    = 1000* sum [1..n]
   |n  == 31   = 1000* sum [1..n]
   |n  > 17    = 500 * n ^ 3
   |n  == 17   = 500 * n ^ 3
   |otherwise = product [1..n]
```

```
points n
  |n <= 16            = product [1..n]
  |16 < n && n <= 30= 500 * n ^ 3
  |31 < n              = 10^3*sum [1..n]
```

STYLE MATTERS!

Haskell Platform



Quiz

```
mystery ::Int->Int->Int->Bool
mystery x y z
   |x+y == z= True
   |x+z == y= True
   |z+y == x= True
   |otherwise= False
```

a) What is

- `mystery 2 3 4`
- `mystery -2 2 0`
- `mystery  10 2 8`

b) Write the function definition without using guards.

Naming requirements

☑ `myFun` `fun1` `arg_2` `x'` `_manyS`

☒ `5days` `Day` ` `` ` `my$s` `where`

case, class, data, default, deriving, do, else, if, import, in, infix, infixl, infixr, instance, let, module, newtype, of, then, type, where

`myName` or `my_name` ?

if..then..else

Example:

```
largest  ::   Int  ->  Int  ->   Int
largest m  n  =  if n >= m then n else m
```

if..then..else can be nested:

```
if   cond1    then r1
else if cond2 then r2
else if cond3 then r3
…
else   rn
```

if..then..else

Fill in the definition of the function `points`

Number N	Points
under 16	N!
between 17 and 30	$500N^3$
between 31 and 50	$10^3 \ \Sigma_{i=1,\ N}\ i$

```
points n =  if                  then
            else                then
           else
```

always

if..then..else

```
verStyle n =  if (n < 10) then 2*n+1
              else if (n >=10 && n <= 50) then n +15
              else if (n > 50 && n <= 80) then n `div` 50
              else  10 * gcd 35 n
```

What can be improved here?

When you see red!

GHCi

```
GHCi, version 7.8.3: http://www.haskell.org/ghc/  :? for help
Loading package ghc-prim ... linking ... done.
Loading package integer-gmp ... linking ... done.
Loading package base ... linking ... done.
Prelude> 5* -9

<interactive>:2:1:
    Precedence parsing error
        cannot mix `*' [infixl 7] and prefix `-' [infixl 6] in the same infix ex
pression
Prelude>
```

WinGHCi

File Edit Actions Tools Help

```
Prelude> 5 * -9

<interactive>:1:1:
    Precedence parsing error
        cannot mix `*' [infixl 7] and prefix `-' [infixl 6] in the same infix expression
Prelude> 5* (-9)
-45
Prelude> abs -4

<interactive>:1:6:
    No instance for (Num (a0 -> a0))
      arising from the literal `4'
    Possible fix: add an instance declaration for (Num (a0 -> a0))
    In the second argument of `(-)', namely `4'
    In the expression: abs - 4
    In an equation for `it': it = abs - 4
Prelude> abs (-4)
4
Prelude>
```

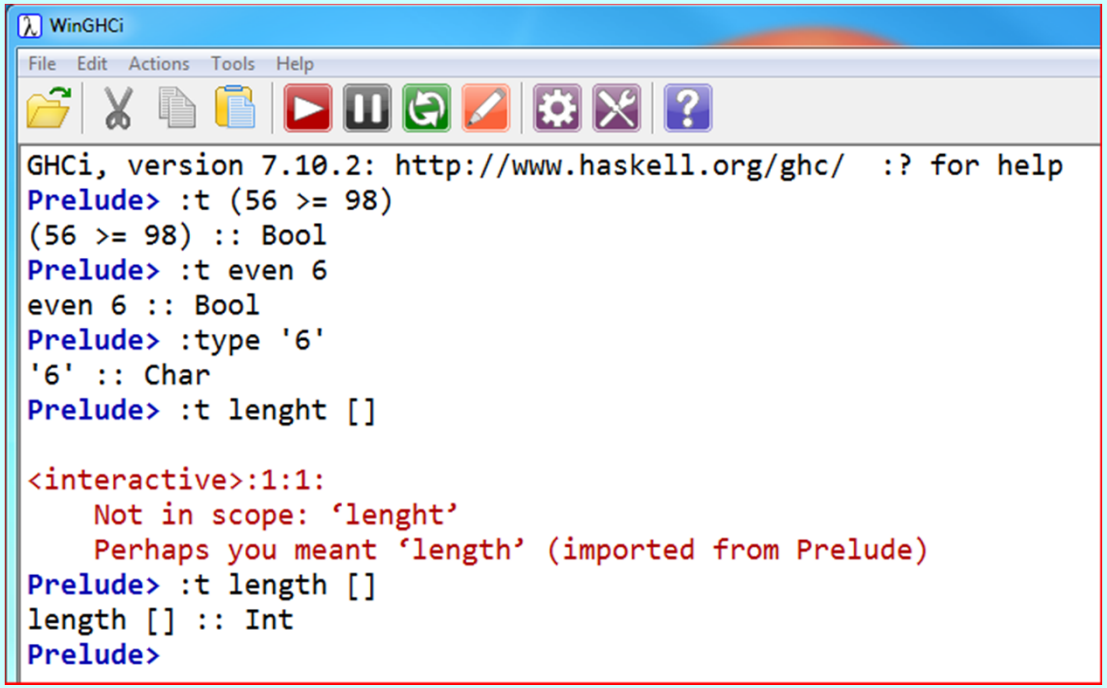
Error function

```
points :: Int -> Int
points n
     |n <= 0|| n > 50= error “Number is not between 1 and 50”
```

Number N	Points
under 16	N!
between 17 and 30	$500N^3$
between 31 and 50	$10^3 \Sigma_{i=1, N} i$

Type inference

The command
`:type <expression>`
asks Haskell to print the type of expression.



More about types

Haskell

- is strongly typed
- is statically typed
- provides type inference

What are the benefits/drawbacks for each?

Two types of scripts

```
-- first.hs
-- function to find the last digit of an integer,
-- assumes  a positive integer.
lastDigit :: Int  ->  Int
lastDigit n  =  n `mod`10
```

```
first.lhs
function to find the last digit of an integer,
assumes a positive integer.

>lastDigit :: Int  ->  Int
>lastDigit n  =  n `mod`10
```

Guards

General form

```
functionName   p1    p2   …   pn
             |  g1            = e1
             |  g2            = e2
             …
             |  otherwise     = e
```

In what order are the guards checked?
What are the possible values of a guard ?

Five steps in writing functions

1. Write a function description (add as a comment).
2. Produce example invocations (optional).
3. Define its type.
4. Define the function equations.
5. Test the function.

Quiz

Write a Haskell function that computes the maximum of three integers

a) Using guards and without using any Haskell function.
b) Using if..then..else
c) Using a single expression --using Haskell function(s).

Summary

- Haskell is a strongly typed functional programming language.
- A Haskell program
  - consists in writing functions
  - is written in a script

- Guards and if then else constructs can be used to write functions.
- Haskell provides various predefined function and operation for working with integers.

Next: Pattern matching and recursion