**Basic coding in Haskell**

Congratulations, You are in GHCI!. The Prompt here is **Prelude>.** You can change it just by typing :set prompt “haskell>”.

Basic Syntax :

**A: Simple Arithmetic example:**

haskell> 2 + 10

12

haskell> 2 \* 10

20

haskell> 10 - 2

8

haskell> 10 / 2

5

**B: We can also use several operations on one line and all the usual precedence rules are obeyed.**

haskell> (50 \* 100) - 4999

1

haskell> 50 \* 100 - 4999

1

haskell> 10 / 2 + 3

8

**C: Boolean algebra is also straightforward.**

haskell> True && False

False

haskell> True && True

True

haskell> False || True

True

haskell> not False

True

**D: Testing Equality**

haskell> 5 == 5

True

haskell> 1 == 0

False

haskell> 5 /= 4

True

haskell> “hello” == “hello”

True

**E: Haskell functions are called by writing the Function name**

haskell> succ 8

9

haskell> min 9 10

9

haskell> max 100 101

101

haskell> succ 9 + max 5 4 + 1

16

**F: Custom function**

haskell> doubleMe x = x + x

haskell>double 9

18

haskell> doubleUs x y = x\*2 + y\*2

haskell>double 4 9

26

haskell>double 2.3 34.2

73.0

**Lists**

haskell> let lostNum = [4,8,15,16,23,42]

haskell> lostNum

[4,8,15,16,23,42]

**Signature of a function**

**Crete a hs file then**

square :: Int -> Int

square v = v \* v

load the file and call the square method

prod :: Int -> Int -> Int

prod x y = x \* y

volumeOfACylinder r h = pi \* r^2 \* h

## Common Data types

### **Integer number types: Int and Integer**

2^62 :: Int

### **Floating-point number types: Float and Double**

3.14159265358979323846 :: Float

3.14159265358979323846 :: Double

### **Boolean type Bool**

usual comparison operators to produce a Bool value:

==,  /=,  <=,  >=,  <,  >

### **Character type Char**

represent a *Unicode* character.

### **Tuples**

('a', 3, True)

# Conditions and helper constructions

## If-then-else expressions

checkLocalhost **::** String **->** String

checkLocalhost ip **=**

*-- True or False?*

**if** ip == "127.0.0.1"

*-- When the condition is True the answer is*

**then** "It's localhost!"

*-- Otherwise, the condition is False and the answer is*

**else** "No, it's not localhost."

checkLocalhost "127.0.0.1"

specialBirthday **::** Int **->** [Char]

specialBirthday age **=**

**if** age == 1

**then** "First birthday!"

**else**

**if** age == 18

**then** "You're an adult!"

**else**

**if** age == 60

**then** "Finally, I can stop caring about new lingo!"

**else** "Nothing special"

**Why Not:**

specialBirthday age

| age == 1 = "First birthday!"

| age == 18 = "You're an adult!"

| age == 60 = "Finally, I can stop caring about new lingo!"

| otherwise = "Nothing special"

specialBirthday **::** Int **->** [Char]

specialBirthday age

| age == 1 **=** "First birthday!"

| age == 18 **=** "You're an adult!"

| age == 60 **=** "Finally, I can stop caring about new lingo!"

| otherwise **=** "Nothing special"

### let expressions

hotterInKelvin' **::** Double **->** Double **->** Double

hotterInKelvin' c f **=**

**let** fToC t **=** (t - 32) \* 5 / 9

cToK t **=** t + 273.16

fToK t **=** cToK (fToC t)

**in** **if** c > fToC f **then** cToK c **else** fToK f

hotterInKelvin' 40 100

**ex-1: sum.hs**

sum' :: Num p => [p] -> p

sum' [] = 0

sum' (n:ns) = n + sum' ns

load and call the method

narendra>:l sum.hs

[1 of 1] Compiling Main ( sum.hs, interpreted )

Ok, one module loaded.

narendra>sum' [1,2,3,4,5,98,34,90]

237

**ex-2: product. Hs**

module Product

( product'

) where

product' :: Num p => [p] -> p

product' [] = 1

product' (x:xs) = x \* product' xs

load and call the method

narendra>:l product.hs

[1 of 1] Compiling Product ( product.hs, interpreted )

Ok, one module loaded.

narendra>product' [1,2,3,4,5,98,34,90]

35985600

**ex-3: ex3.hs**

import Product

main = do

print (product' [2, 3, 4])

load ad call the method

narendra>:l ex3.hs product.hs

[1 of 2] Compiling Product ( product.hs, interpreted )

[2 of 2] Compiling Main ( ex3.hs, interpreted )

Ok, two modules loaded.

narendra>main

24