# Lecture 8 – Randomness

Violet Ka I Pun

violet@ifi.uio.no

# Randomness in Haskell

Haskell is pure

- Cannot change the state of a program
- No side-effect
  - E.g.: A function always returns the same value with the same parameters
- How to get random values in the pure Haskell?

# Randomness in Haskell – System.Random

```
random :: (Random a, RandomGen g) => g -> (a, g)
```

- Random: for values which can be produced by a random number generator

  E.g.: `Float`, `Int`, `Bool` are instances of `Random`

- RandomGen: a generator for a random sequence of values

E.g.: `mkStdGen :: Int -> StdGen`

StdGen is an instance of `RandomGen`

# Generate some random values

```
random :: (Random a, RandomGen g) => g -> (a, g)
```

An example:

```
random (mkStdGen 9) in GHCi .....................................error
```

Have to specify the type of random values!!!

E.g.,

```
random (mkStdGen 9) :: (Int, StdGen)
.........................(7636710457939723046,578976272 2103410263)
```

What about running `random` for the second time with

– differen type?

```
random (mkStdGen 9) :: (Float, StdGen)
.................................(0.8038231,978817019 1655838864)
```

– different parameter?

```
random (mkStdGen 18) :: (Int, StdGen)
.....................(-3056066145809144726,1959048342 2103410263)
```

# Multiple random values with same paramenter

What about calling `random` with the same parameter twice?

1st
```
random (mkStdGen 9) :: (Int, StdGen)
.....(763671045793972304 6,578976272 2103410263)
```

2nd
```
random (mkStdGen 9) :: (Int, StdGen)
.....(763671045793972304 6,578976272 2103410263)
```

Same result!!! Of course, Haskell is has no side-effects!

# Multiple random values

How to generate multiple random values with one single parameter?

```
random :: (Random a, RandomGen g) => g -> (a, g)
```

E.g., generate three random integers:

```
gen3Int :: StdGen -> (Int, Int, Int)
gen3Int gen =
   let (first, gen1) = random gen
       (second, gen2) = random gen1
       (third, gen3) = random gen2
   in (first, second, third)
```

or:

```
gen3Int' :: Int -> (Int, Int, Int)
gen3Int' n =
   let gen = mkStdGen n
       (first, gen1) = random gen
       (second, gen2) = random gen1
       (third, gen3) = random gen2
   in (first, second, third)
```

**Generate 100 integers?**

# Multiple random values

```
randoms :: (Random a, RandomGen g) => g -> [a]
```

$\rightarrow$ returns a an infinite list of random values

```
randoms (mkStdGen 8) :: [Int]
....................[-3985753702595628 70,-6370604356117182359, ...]
```

3 random integers:

```
take 3 $ randoms (mkStdGen 8) :: [Int]
...[-3985753702595628 70,-6370604356117182359,8399777519602674086]
```

5 random boolean values:

```
take 5 $ randoms (mkStdGen 88) :: [Bool]
.....................................[True,True,False,False,False]
```

# Random values within a range

```
randomR :: (Random a, RandomGen g) => (a, a) -> g -> (a, g)

randomR (1,10) (mkStdGen 201)
.....................................................(6,8082828 40692)

randomR ('a','z') (mkStdGen 100) :: (Char, StdGen)
.....................................................('x',4041414 40692)

randomRs :: (Random a, RandomGen g) => (a, a) -> g -> [a]
```

→returns an infinite list of random values within a given range

6 random integers between 1 and 50

```
take 6 $ randomRs (1,50) (mkStdGen 34) :: [Int]
.....................................................[48,20,1,34,30,10]
```

10 random characters between 'a' and 'z'

```
take 10 $ randomRs ('a','z') (mkStdGen 30) :: [Char]
......................................................"xkmiqttnvu"
```

# Do we really have some random values?

So far we just manually make our "random number generator", with some arbitrary numbers.

- it will always return the same random number given the same arbitrary number.

## Global generator

- A random number generator from the system when the program starts
- `getStdGen :: IO StdGen` ..... an IO action!!

```
main = do
  gen <- getStdGen
  putStr $ take 10 (randomRs ('a','z') gen)
  gen' <- getStdGen
  putStr $ take 10 (randomRs ('a','z') gen')
```

- Produce a different string each time the function is called.
- Print the same string twice

# New global generator

```
newStdGen :: IO StdGen
```

- Split the current global random generator into two
- Update the current global one with one of the two
- Return the other

```
main = do
  gen <- getStdGen
  putStr $ take 10 (randomRs ('a','z') gen)
  gen' <- getStdGennewStdGen
  putStr $ take 10 (randomRs ('a','z') gen')
```

# Guess a word

```
main = do
  gen <- getStdGennewStdGen
  putStrLn "Input a sequence of words separated by space:"
  l <- getLine
  let l' = words l
      n = length l'
      (pos, newGen) = randomR (0,n-1) gen :: (Int, StdGen)
      picked = l' !! pos
  putStrLn " Guess which word I pick from the list:"
  w <- getLine
  if (w == picked)
  then putStrLn "Correct!"
  else putStrLn $ No, it was "++ picked
  main
```

# Read more in

Learn You a Haskell for Great Good!

Chapter 9

http://learnyouahaskell.com/input-and-output