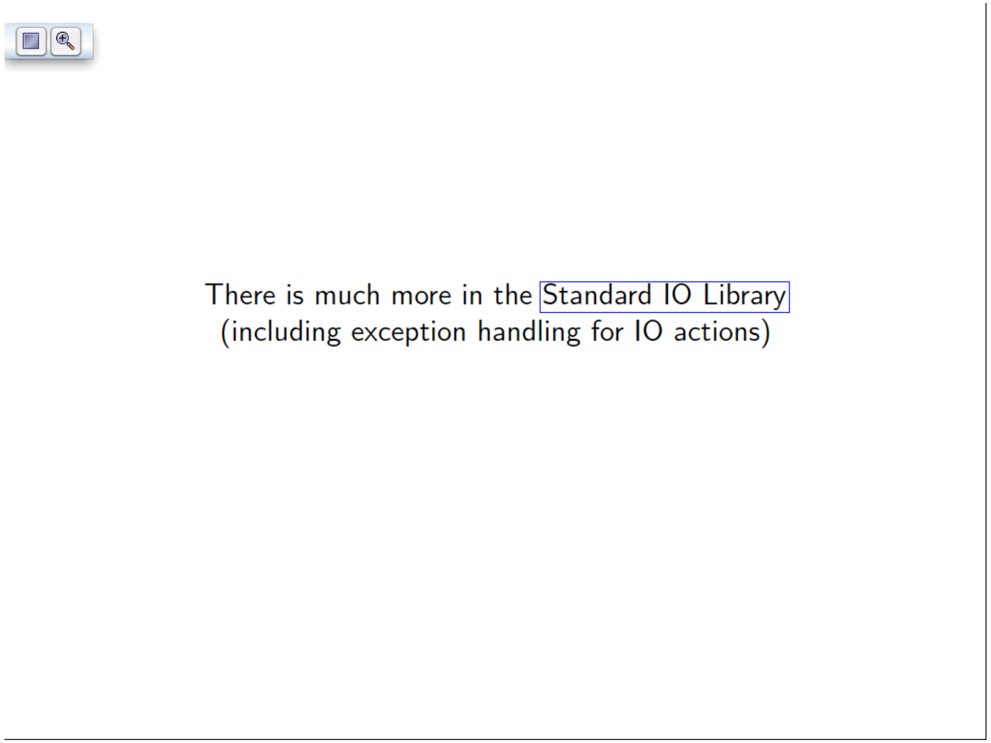
Script generated by TTT

Title: Nipkow: Info2 (12.12.2014)

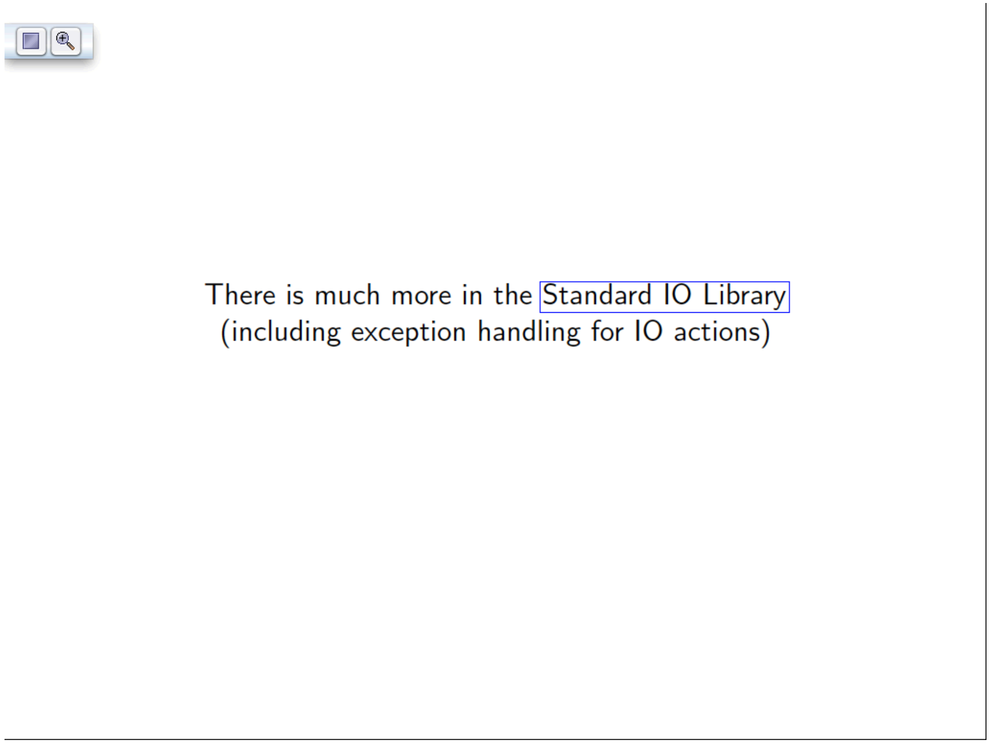
Date: Fri Dec 12 09:01:53 CET 2014

Duration: 48:46 min

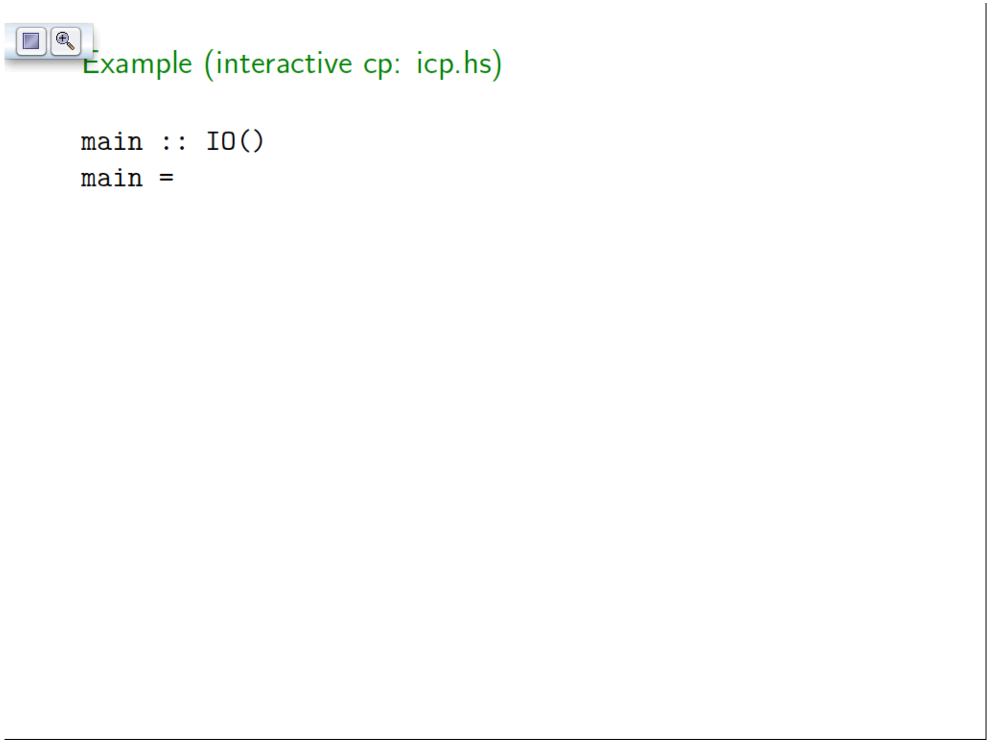
Pages: 104



There is much more in the Standard IO Library
(including exception handling for IO actions)



There is much more in the Standard IO Library
(including exception handling for IO actions)



Example (interactive cp: icp.hs)

```
main :: IO()
main =
```

Example (interactive cp: icp.hs)

```
main :: IO()
main =
  do fromH <- readOpenFile "Copy from: " ReadMode
```

Example (interactive cp: icp.hs)

```
main :: IO()
main =
  do fromH <- readOpenFile "Copy from: " ReadMode
     toH <- readOpenFile "Copy to: " WriteMode
```

Example (interactive cp: icp.hs)

```
main :: IO()
main =
  do fromH <- readOpenFile "Copy from: " ReadMode
     toH <- readOpenFile "Copy to: " WriteMode
     contents <- hGetContents fromH
```

Example (interactive cp: icp.hs)

```
main :: IO()
main =
  do fromH <- readOpenFile "Copy from: " ReadMode
     toH <- readOpenFile "Copy to: " WriteMode
     contents <- hGetContents fromH
     hPutStr toH contents
     hClose fromH
     hClose toH
```

Example (interactive cp: icp.hs)

```
main :: IO()
main =
  do fromH <- readOpenFile "Copy from: " ReadMode
     toH <- readOpenFile "Copy to: " WriteMode
     contents <- hGetContents fromH
     hPutStr toH contents
     hClose fromH
     hClose toH

readOpenFile :: String -> IOMode -> IO Handle
readOpenFile prompt mode =
  do putStrLn prompt
     name <- getLine
```

Example (interactive cp: icp.hs)

```
main :: IO()
main =
  do fromH <- readOpenFile "Copy from: " ReadMode
     toH <- readOpenFile "Copy to: " WriteMode
     contents <- hGetContents fromH
     hPutStr toH contents
     hClose fromH
     hClose toH

readOpenFile :: String -> IOMode -> IO Handle
readOpenFile prompt mode =
  do putStrLn prompt
     name <- getLine
     handle <- openFile name mode
```

Executing xyz.hs

If xyz.hs contains a definition of main:

- runhaskell xyz

Example (interactive cp: icp.hs)

```
main :: IO()
main =
  do fromH <- readOpenFile "Copy from: " ReadMode
     toH <- readOpenFile "Copy to: " WriteMode
     contents <- hGetContents fromH
     hPutStr toH contents
     hClose fromH
     hClose toH

readOpenFile :: String -> IOMode -> IO Handle
readOpenFile prompt mode =
  do putStrLn prompt
     name <- getLine
     handle <- openFile name mode
     return handle
```

There is much more in the Standard IO Library
(including exception handling for IO actions)

Executing `xyz.hs`

If `xyz.hs` contains a definition of `main`:

- `runhaskell xyz`

Executing `xyz.hs`

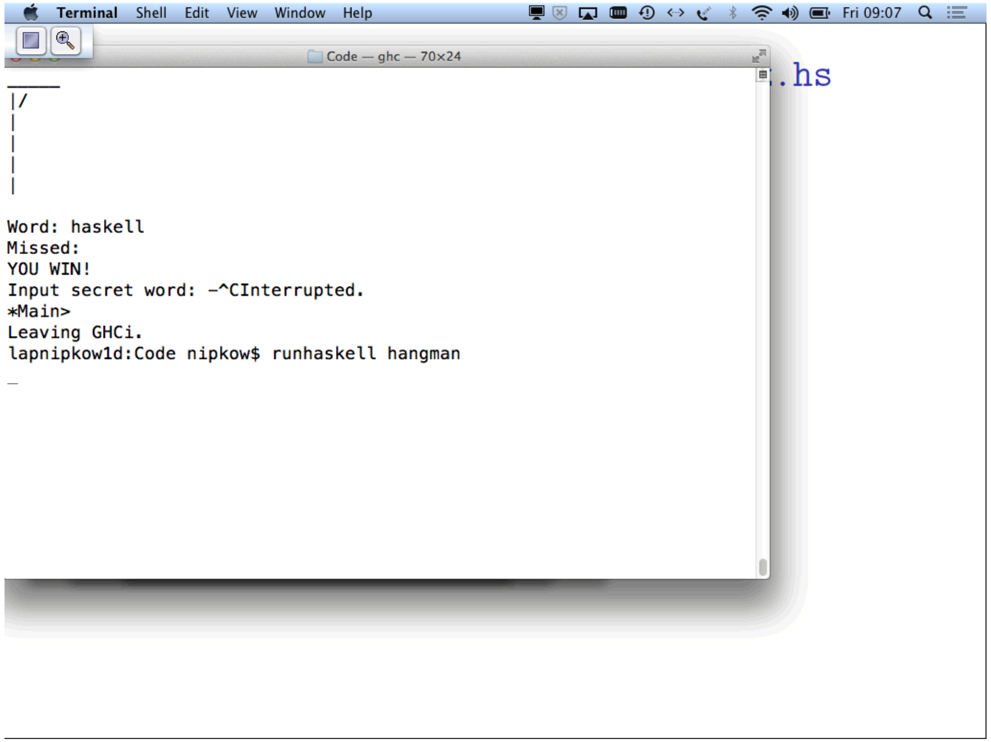
If `xyz.hs` contains a definition of `main`:

- `runhaskell xyz`
  or
- `ghc xyz`

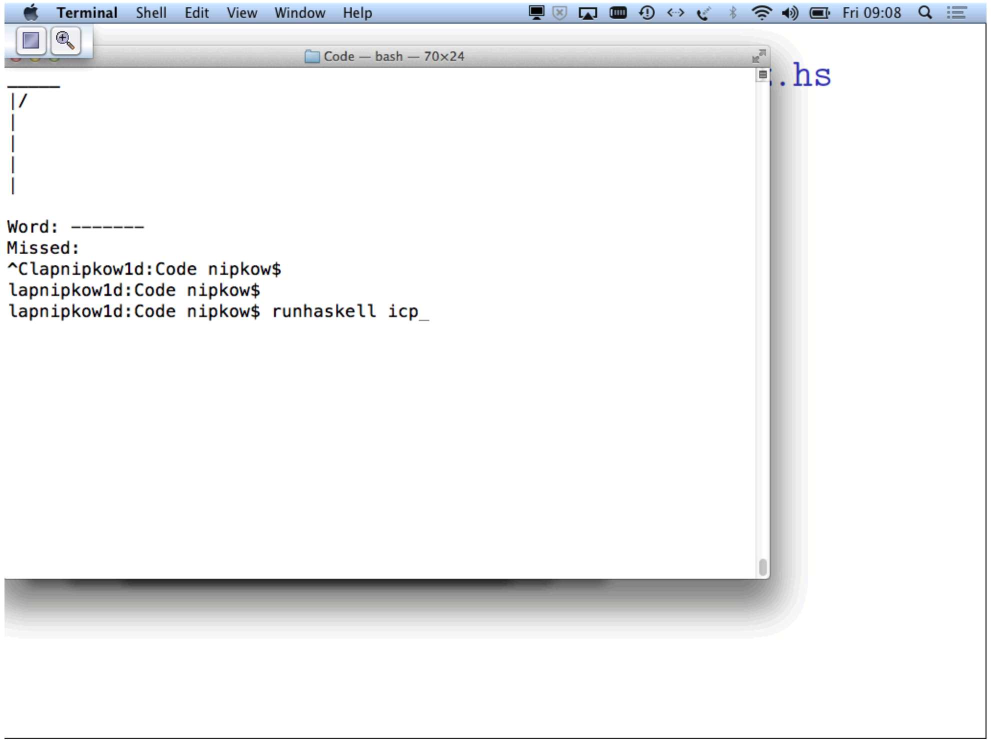
Executing `xyz.hs`

If `xyz.hs` contains a definition of `main`:

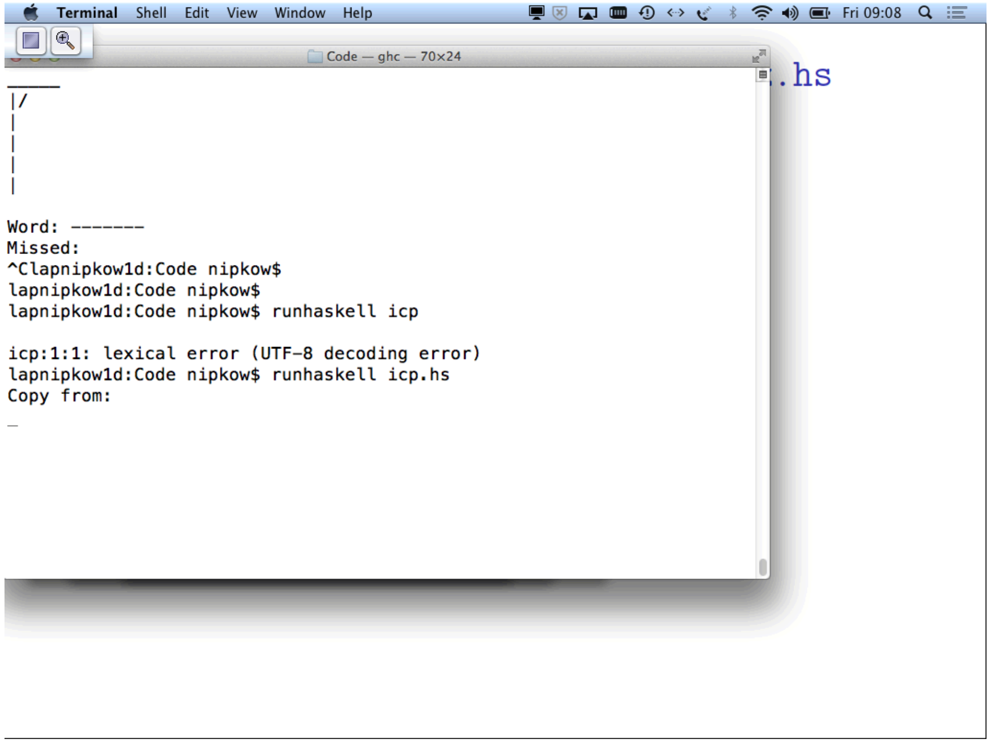
- `runhaskell xyz`
  or
- `ghc xyz` ⇝ executable file `xyz`



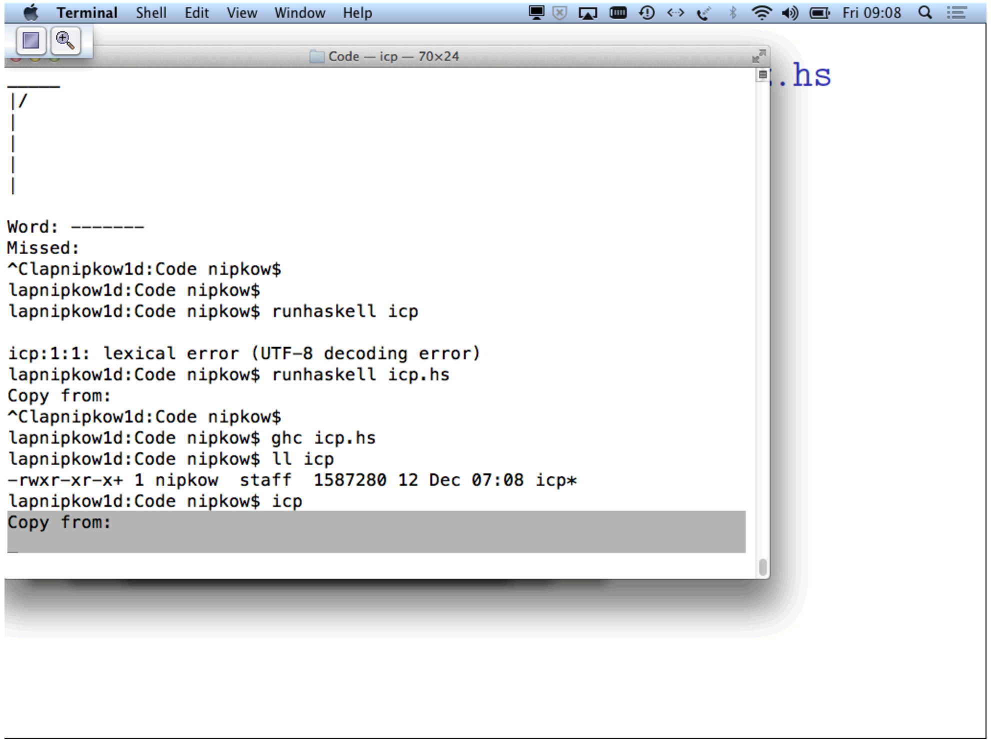
```
_____
|/
|
|
|
|


Word: haskell
Missed:
YOU WIN!
Input secret word: -^CInterrupted.
*Main>
Leaving GHCi.
lapnipkow1d:Code nipkow$ runhaskell hangman

_
```



```
_____
|/
|
|
|
|


Word: -------
Missed:
^Clapnipkow1d:Code nipkow$
lapnipkow1d:Code nipkow$
lapnipkow1d:Code nipkow$ runhaskell icp_
```



```
_____
|/
|
|
|
|


Word: -------
Missed:
^Clapnipkow1d:Code nipkow$
lapnipkow1d:Code nipkow$
lapnipkow1d:Code nipkow$ runhaskell icp

icp:1:1: lexical error (UTF-8 decoding error)
lapnipkow1d:Code nipkow$ runhaskell icp.hs
Copy from:

_
```



```
_____
|/
|
|
|
|


Word: -------
Missed:
^Clapnipkow1d:Code nipkow$
lapnipkow1d:Code nipkow$
lapnipkow1d:Code nipkow$ runhaskell icp

icp:1:1: lexical error (UTF-8 decoding error)
lapnipkow1d:Code nipkow$ runhaskell icp.hs
Copy from:
^Clapnipkow1d:Code nipkow$
lapnipkow1d:Code nipkow$ ghc icp.hs
lapnipkow1d:Code nipkow$ ll icp
-rwxr-xr-x+ 1 nipkow  staff  1587280 12 Dec 07:08 icp*
lapnipkow1d:Code nipkow$ icp
Copy from:
```

Executing `xyz.hs`

If `xyz.hs` contains a definition of `main`:

- `runhaskell xyz`
  or
- `ghc xyz` ⇝ executable file `xyz`

9.2 Network I/O

Types

Types

- `data Socket`
  A socket is one endpoint of a two-way communication link between two programs running on the network.

Types

- `data Socket`
  A socket is one endpoint of a two-way communication link between two programs running on the network.
- `data PortId = PortNumber PortNumber | ...`

Types

- `data Socket`
  A socket is one endpoint of a two-way communication link between two programs running on the network.
- `data PortId = PortNumber PortNumber | ...`
- `data PortNumber`

Types

- `data Socket`
  A socket is one endpoint of a two-way communication link between two programs running on the network.
- `data PortId = PortNumber PortNumber | ...`
- `data PortNumber`
  `instance Num PortNumber`

Types

- `data Socket`
  A socket is one endpoint of a two-way communication link between two programs running on the network.
- `data PortId = PortNumber PortNumber | ...`
- `data PortNumber`
  `instance Num PortNumber`
  ⟹ `PortNumber 9000 :: PortId`

Server functions

Server functions

- `listenOn :: PortId -> IO Socket`
  Create server side socket for specific port

Server functions

- `listenOn :: PortId -> IO Socket`
  Create server side socket for specific port
- `accept :: Socket -> IO (Handle, ..., ...)`
  $\Longrightarrow$ can read/write from/to socket via handle

Server functions

- `listenOn :: PortId -> IO Socket`
  Create server side socket for specific port
- `accept :: Socket -> IO (Handle, ..., ...)`
  $\Longrightarrow$ can read/write from/to socket via handle
- `sClose :: Socket -> IO ()`
  Close socket

Initialization for Windows

```
withSocketsDo :: IO a -> IO a
```

Initialization for Windows

```
withSocketsDo :: IO a -> IO a
```

Standard use pattern:

```
main  =  withSocketsDo $ do ...
```

Example (pingPong.hs)

Example (pingPong.hs)

```
main :: IO ()
main  =  withSocketsDo $ do
```

Example (pingPong.hs)

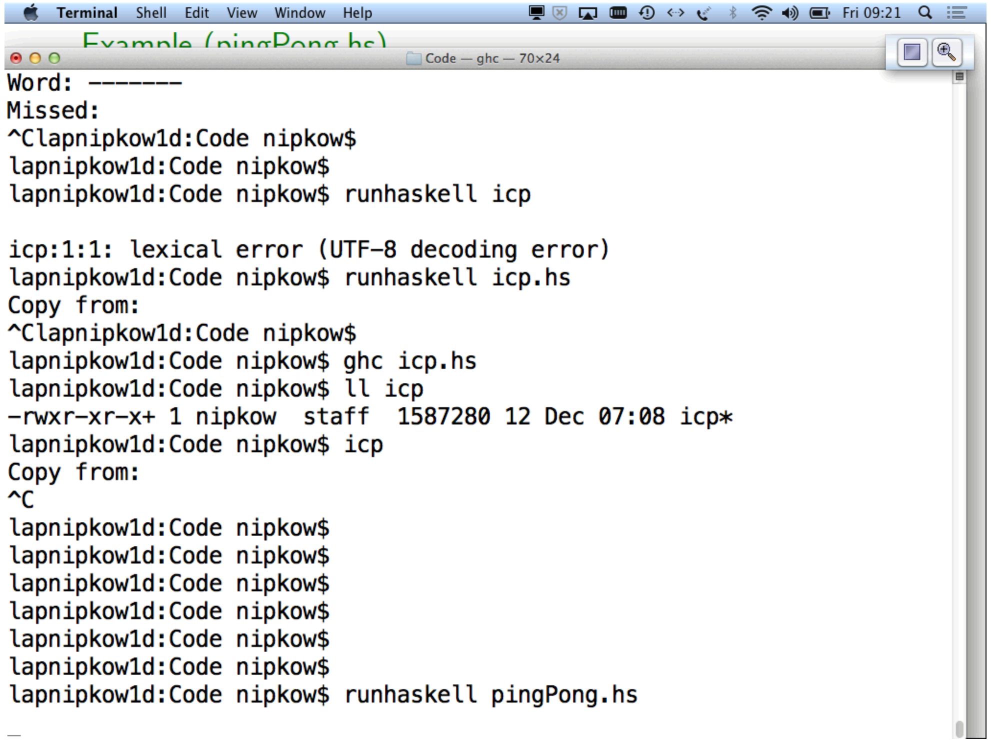
```
main :: IO ()
main  =  withSocketsDo $ do
  sock <- listenOn $ PortNumber 9000
  (h, _, _) <- accept sock
```

Example (pingPong.hs)

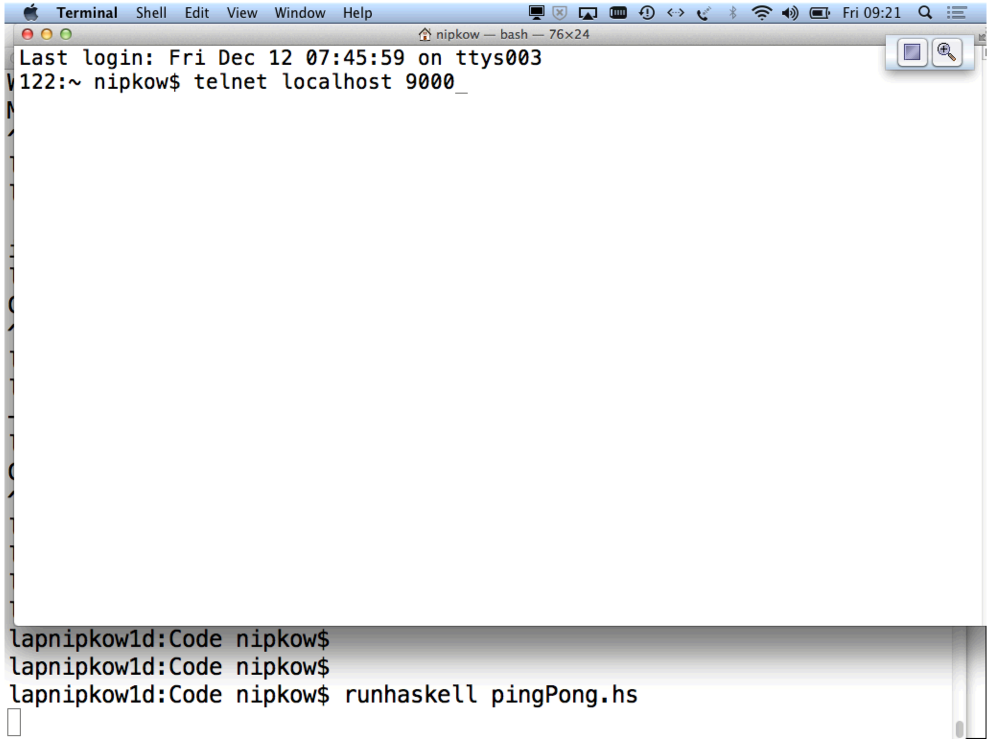
```
main :: IO ()
main  =  withSocketsDo $ do
  sock <- listenOn $ PortNumber 9000
  (h, _, _) <- accept sock
  hSetBuffering h LineBuffering
```

Example (pingPong.hs)

```
main :: IO ()
main  =  withSocketsDo $ do
  sock <- listenOn $ PortNumber 9000
  (h, _, _) <- accept sock
  hSetBuffering h LineBuffering
  loop h
  sClose sock

loop :: Handle -> IO ()
loop h  =  do
  input <- hGetLine h
  if take 4 input == "quit"
  then do hPutStrLn h "goodbye!"
          hClose h
  else do hPutStrLn h ("got " ++ input)
          loop h
```

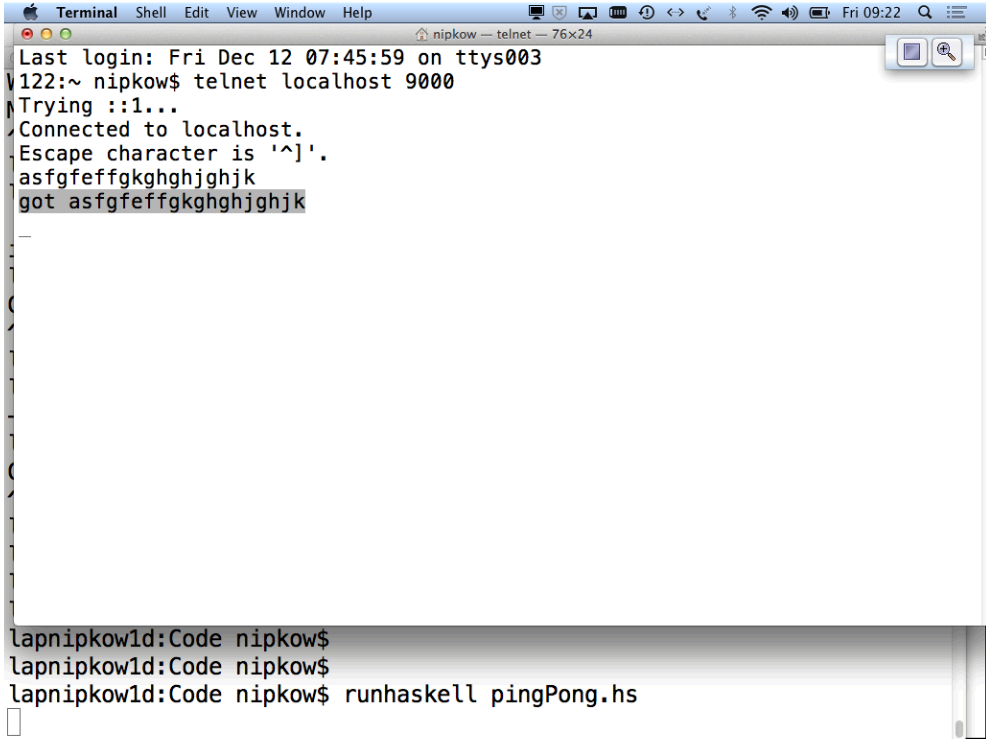


```
Word: --------
Missed:
^Clapnipkow1d:Code nipkow$
lapnipkow1d:Code nipkow$
lapnipkow1d:Code nipkow$ runhaskell icp

icp:1:1: lexical error (UTF-8 decoding error)
lapnipkow1d:Code nipkow$ runhaskell icp.hs
Copy from:
^Clapnipkow1d:Code nipkow$
lapnipkow1d:Code nipkow$ ghc icp.hs
lapnipkow1d:Code nipkow$ ll icp
-rwxr-xr-x+ 1 nipkow  staff  1587280 12 Dec 07:08 icp*
lapnipkow1d:Code nipkow$ icp
Copy from:
^C
lapnipkow1d:Code nipkow$
lapnipkow1d:Code nipkow$
lapnipkow1d:Code nipkow$
lapnipkow1d:Code nipkow$
lapnipkow1d:Code nipkow$
lapnipkow1d:Code nipkow$
lapnipkow1d:Code nipkow$ runhaskell pingPong.hs
```



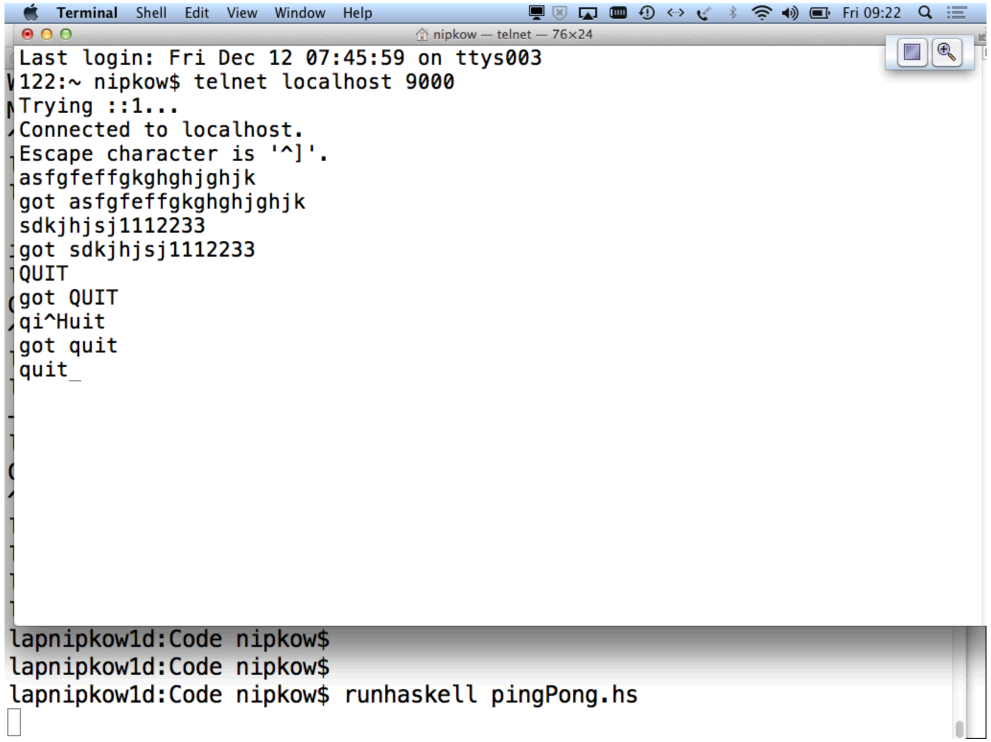
```
Last login: Fri Dec 12 07:45:59 on ttys003
122:~ nipkow$ telnet localhost 9000
```

```
lapnipkow1d:Code nipkow$
lapnipkow1d:Code nipkow$
lapnipkow1d:Code nipkow$ runhaskell pingPong.hs
```



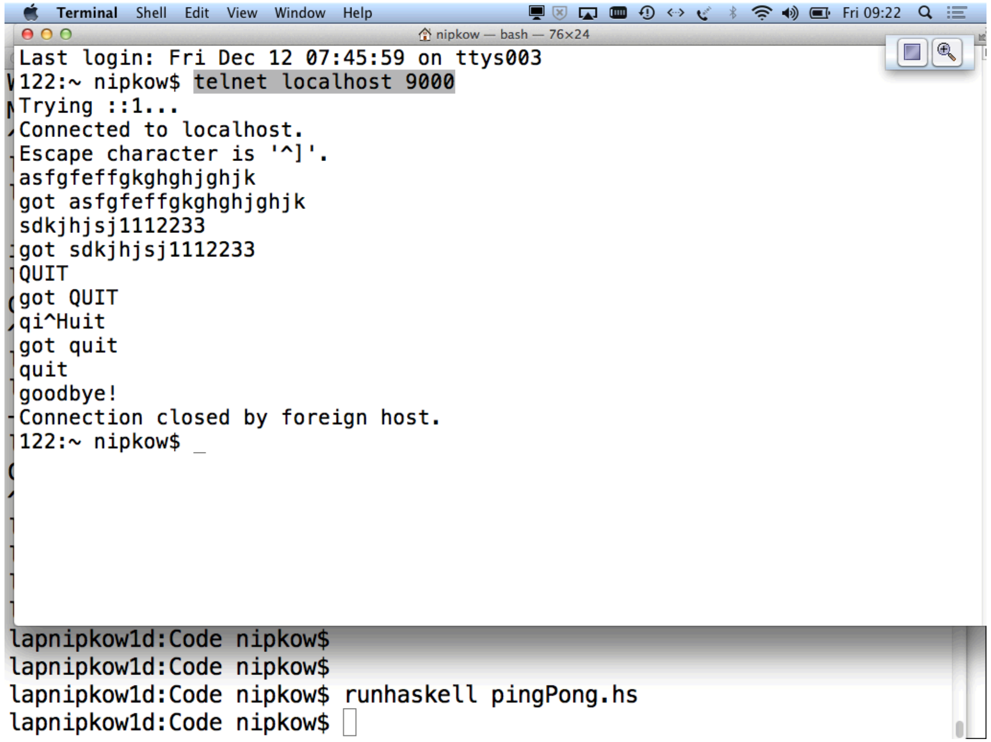
```
Last login: Fri Dec 12 07:45:59 on ttys003
122:~ nipkow$ telnet localhost 9000
Trying ::1...
Connected to localhost.
Escape character is '^]'.
asfgfeffgkghghjghjk
got asfgfeffgkghghjghjk
```

```
lapnipkow1d:Code nipkow$
lapnipkow1d:Code nipkow$
lapnipkow1d:Code nipkow$ runhaskell pingPong.hs
```



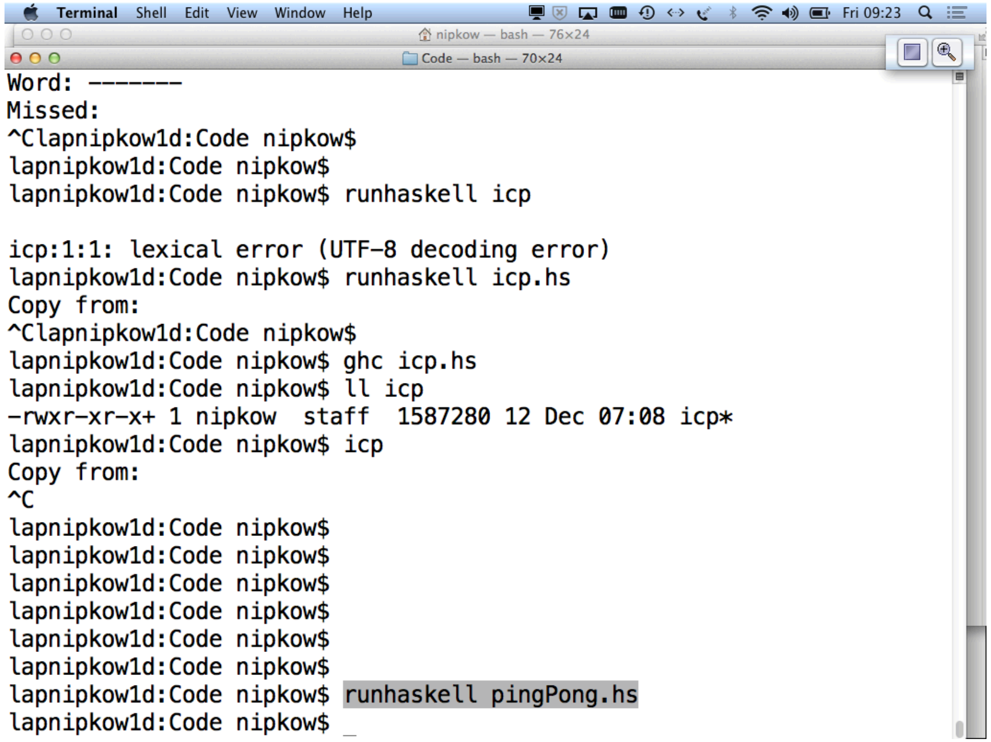
```
Last login: Fri Dec 12 07:45:59 on ttys003
122:~ nipkow$ telnet localhost 9000
Trying ::1...
Connected to localhost.
Escape character is '^]'.
asfgfeffgkghghjghjk
got asfgfeffgkghghjghjk
sdkjhjsj1112233
got sdkjhjsj1112233
QUIT
got QUIT
qi^Huit
got quit
quit
```

```
lapnipkow1d:Code nipkow$
lapnipkow1d:Code nipkow$
lapnipkow1d:Code nipkow$ runhaskell pingPong.hs
```



```
Last login: Fri Dec 12 07:45:59 on ttys003
122:~ nipkow$ telnet localhost 9000
Trying ::1...
Connected to localhost.
Escape character is '^]'.
asfgfeffgkghghjghjk
got asfgfeffgkghghjghjk
sdkjhjsj1112233
got sdkjhjsj1112233
QUIT
got QUIT
qi^Huit
got quit
quit
goodbye!
Connection closed by foreign host.
122:~ nipkow$
```

```
lapnipkow1d:Code nipkow$
lapnipkow1d:Code nipkow$
lapnipkow1d:Code nipkow$ runhaskell pingPong.hs
lapnipkow1d:Code nipkow$
```



```
Word: -------
Missed:
^Clapnipkow1d:Code nipkow$
lapnipkow1d:Code nipkow$
lapnipkow1d:Code nipkow$ runhaskell icp

icp:1:1: lexical error (UTF-8 decoding error)
lapnipkow1d:Code nipkow$ runhaskell icp.hs
Copy from:
^Clapnipkow1d:Code nipkow$
lapnipkow1d:Code nipkow$ ghc icp.hs
lapnipkow1d:Code nipkow$ ll icp
-rwxr-xr-x+ 1 nipkow  staff  1587280 12 Dec 07:08 icp*
lapnipkow1d:Code nipkow$ icp
Copy from:
^C
lapnipkow1d:Code nipkow$
lapnipkow1d:Code nipkow$
lapnipkow1d:Code nipkow$
lapnipkow1d:Code nipkow$
lapnipkow1d:Code nipkow$
lapnipkow1d:Code nipkow$
lapnipkow1d:Code nipkow$ runhaskell pingPong.hs
lapnipkow1d:Code nipkow$ _
```

Example (pingPong.hs)

```
main :: IO ()
main  =  withSocketsDo $ do
  sock <- listenOn $ PortNumber 9000
  (h, _, _) <- accept sock
  hSetBuffering h LineBuffering
  loop h
  sClose sock

loop :: Handle -> IO ()
loop h  =  do
  input <- hGetLine h
  if take 4 input == "quit"
  then do hPutStrLn h "goodbye!"
          hClose h
  else do hPutStrLn h ("got " ++ input)
          loop h
```

Client functions

Client functions

- `type HostName = String`
  For example `"haskell.org"` or `"192.168.0.1"`

Client functions

- `type HostName = String`
  For example `"haskell.org"` or `"192.168.0.1"`

- `connectTo :: HostName -> PortId -> IO Handle`
  Connect to specific port of specific host

Example (wGet.hs)

Example (wGet.hs)

```
main :: IO()
main = withSocketsDo $ do
```

Example (wGet.hs)

```
main :: IO()
main = withSocketsDo $ do
  putStrLn "Host?"
  host <- getLine
  h <- connectTo host (PortNumber 80)
  hSetBuffering h LineBuffering
```

Example (wGet.hs)

```
main :: IO()
main = withSocketsDo $ do
  putStrLn "Host?"
  host <- getLine
  h <- connectTo host (PortNumber 80)
  hSetBuffering h LineBuffering
  putStrLn "Resource?"
  res <- getLine
```

Example (wGet.hs)

```
main :: IO()
main = withSocketsDo $ do
  putStrLn "Host?"
  host <- getLine
  h <- connectTo host (PortNumber 80)
  hSetBuffering h LineBuffering
  putStrLn "Resource?"
  res <- getLine
  hPutStrLn h ("GET " ++ res ++ " HTTP/1.0\n")
```

Example (wGet.hs)

```
main :: IO()
main = withSocketsDo $ do
  putStrLn "Host?"
  host <- getLine
  h <- connectTo host (PortNumber 80)
  hSetBuffering h LineBuffering
  putStrLn "Resource?"
  res <- getLine
  hPutStrLn h ("GET " ++ res ++ " HTTP/1.0\n")
  s <- hGetContents h
```

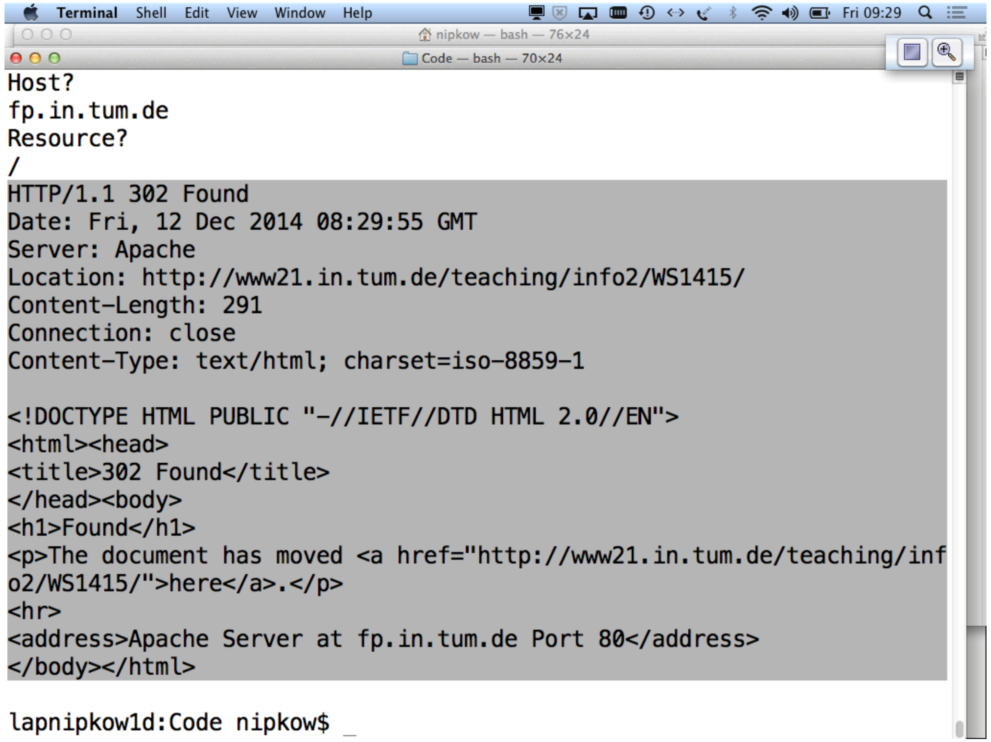
For more detail see

http://hackage.haskell.org/package/network/docs/Network.html

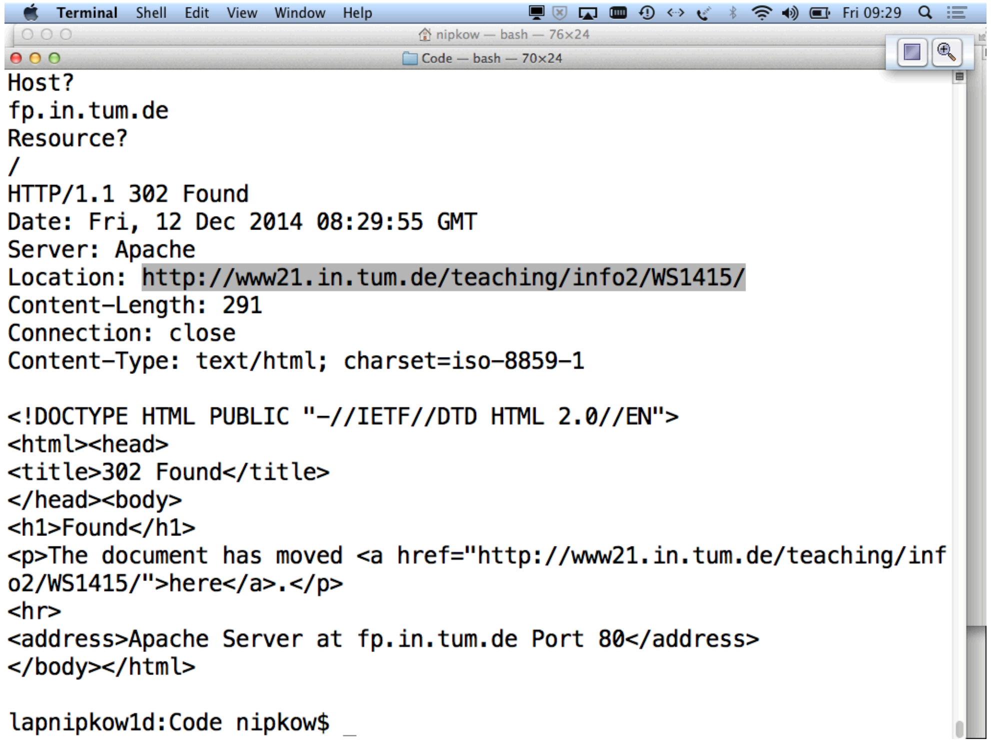
http://hackage.haskell.org/package/network/docs/Network-Socket.html

Example (wGet.hs)

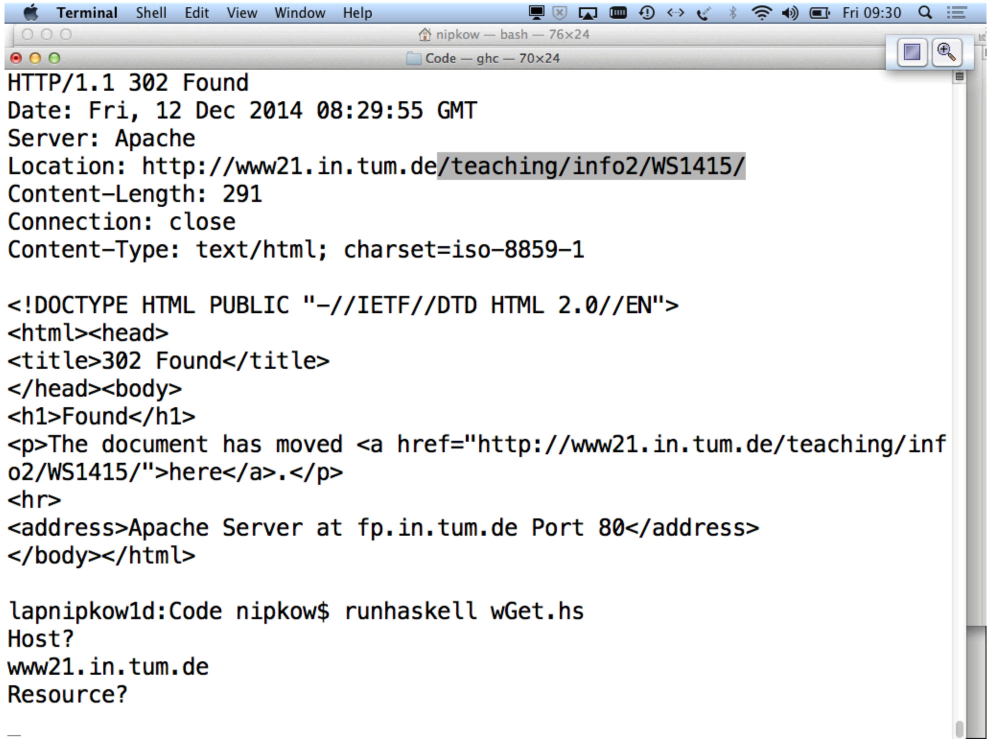
```haskell
main :: IO()
main = withSocketsDo $ do
  putStrLn "Host?"
  host <- getLine
  h <- connectTo host (PortNumber 80)
  hSetBuffering h LineBuffering
  putStrLn "Resource?"
  res <- getLine
  hPutStrLn h ("GET " ++ res ++ " HTTP/1.0\n")
  s <- hGetContents h
  putStrLn s
```



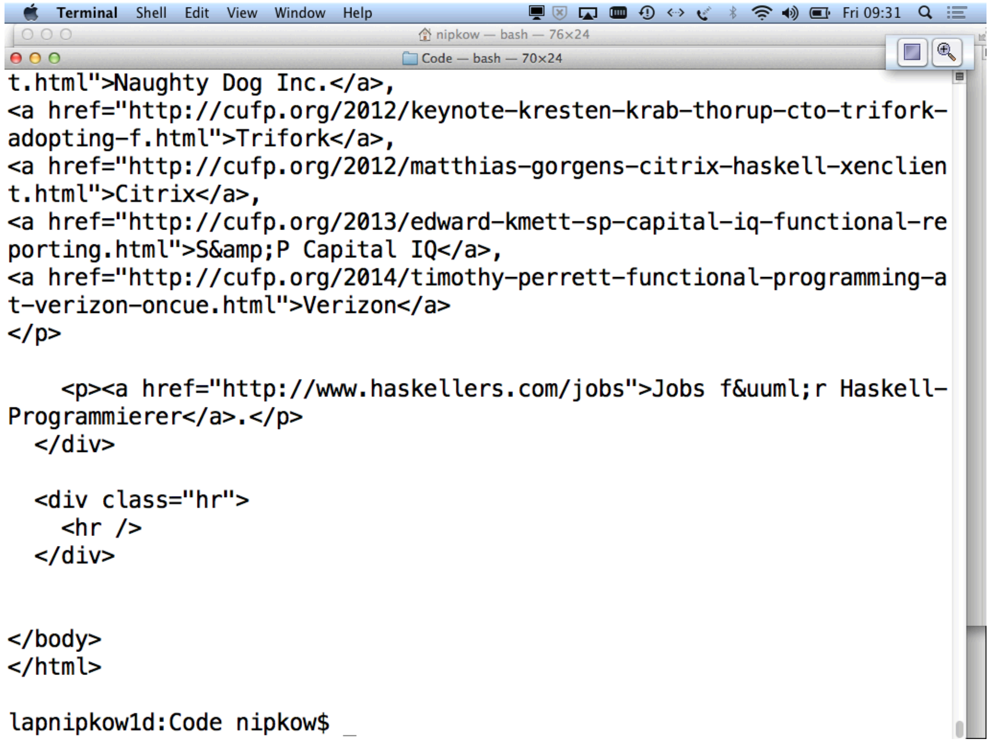
```
Host?
fp.in.tum.de
Resource?
/
HTTP/1.1 302 Found
Date: Fri, 12 Dec 2014 08:29:55 GMT
Server: Apache
Location: http://www21.in.tum.de/teaching/info2/WS1415/
Content-Length: 291
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>302 Found</title>
</head><body>
<h1>Found</h1>
<p>The document has moved <a href="http://www21.in.tum.de/teaching/inf
o2/WS1415/">here</a>.</p>
<hr>
<address>Apache Server at fp.in.tum.de Port 80</address>
</body></html>

lapnipkow1d:Code nipkow$ _
```



```
Host?
fp.in.tum.de
Resource?
/
HTTP/1.1 302 Found
Date: Fri, 12 Dec 2014 08:29:55 GMT
Server: Apache
Location: http://www21.in.tum.de/teaching/info2/WS1415/
Content-Length: 291
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>302 Found</title>
</head><body>
<h1>Found</h1>
<p>The document has moved <a href="http://www21.in.tum.de/teaching/inf
o2/WS1415/">here</a>.</p>
<hr>
<address>Apache Server at fp.in.tum.de Port 80</address>
</body></html>

lapnipkow1d:Code nipkow$ _
```



```
HTTP/1.1 302 Found
Date: Fri, 12 Dec 2014 08:29:55 GMT
Server: Apache
Location: http://www21.in.tum.de/teaching/info2/WS1415/
Content-Length: 291
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>302 Found</title>
</head><body>
<h1>Found</h1>
<p>The document has moved <a href="http://www21.in.tum.de/teaching/inf
o2/WS1415/">here</a>.</p>
<hr>
<address>Apache Server at fp.in.tum.de Port 80</address>
</body></html>

lapnipkow1d:Code nipkow$ runhaskell wGet.hs
Host?
www21.in.tum.de
Resource?
_
```



```
t.html">Naughty Dog Inc.</a>,
<a href="http://cufp.org/2012/keynote-kresten-krab-thorup-cto-trifork-
adopting-f.html">Trifork</a>,
<a href="http://cufp.org/2012/matthias-gorgens-citrix-haskell-xenclien
t.html">Citrix</a>,
<a href="http://cufp.org/2013/edward-kmett-sp-capital-iq-functional-re
porting.html">S&amp;P Capital IQ</a>,
<a href="http://cufp.org/2014/timothy-perrett-functional-programming-a
t-verizon-oncue.html">Verizon</a>
</p>

    <p><a href="http://www.haskellers.com/jobs">Jobs f&uuml;r Haskell-
Programmierer</a>.</p>
  </div>

  <div class="hr">
    <hr />
  </div>


</body>
</html>

lapnipkow1d:Code nipkow$ _
```

Example (wGet.hs)

```
main :: IO()
main = withSocketsDo $ do
  putStrLn "Host?"
  host <- getLine
  h <- connectTo host (PortNumber 80)
  hSetBuffering h LineBuffering
  putStrLn "Resource?"
  res <- getLine
  hPutStrLn h ("GET " ++ res ++ " HTTP/1.0\n")
  s <- hGetContents h
  putStrLn s
```

For more detail see

http://hackage.haskell.org/package/network/docs/Network.html

http://hackage.haskell.org/package/network/docs/Network-Socket.html

10.1 Modules

10.1 Modules

Module = collection of type, function, class etc definitions

Purposes:

- Grouping
- Interfaces
- Division of labour

10.1 Modules

Module = collection of type, function, class etc definitions

Purposes:

- Grouping
- Interfaces
- Division of labour
- Name space management: `M.f` vs `f`

10.1 Modules

Module = collection of type, function, class etc definitions

Purposes:

- Grouping
- Interfaces
- Division of labour
- Name space management: `M.f` vs `f`
- Information hiding

10.1 Modules

Module = collection of type, function, class etc definitions

Purposes:

- Grouping
- Interfaces
- Division of labour
- Name space management: `M.f` vs `f`
- Information hiding

GHC: one module per file

10.1 Modules

Module = collection of type, function, class etc definitions

Purposes:

- Grouping
- Interfaces
- Division of labour
- Name space management: `M.f` vs `f`
- Information hiding

GHC: one module per file

Recommendation: module `M` in file `M.hs`

Module header

```
module M where
```

Module header

```
module M where    -- M must start with capital letter
```

Module header

```
module M where    -- M must start with capital letter
↑
```

All definitions must start in this column

Module header

```
module M where    -- M must start with capital letter
↑
```

All definitions must start in this column

- Exports everything defined in `M` (at the top level)

Module header

```
module M where    -- M must start with capital letter
↑
```

All definitions must start in this column

- Exports everything defined in `M` (at the top level)

Selective export:

```
module M (T, f, ...)  where
```

Exporting data types

Exporting data types

```
module M (T) where
data T = ...
```

Exporting data types

```
module M (T) where
data T = ...
```

- Exports only T, but not its constructors

Types

- `data Socket`
  A socket is one endpoint of a two-way communication link between two programs running on the network.

Exporting data types

```
module M (T) where
data T = ...
```

- Exports only T, but not its constructors

```
module M (T(C,D,...)) where
data T = ...
```

- Exports T and its constructors C, D, ...

Exporting data types

```
module M (T) where
data T = ...
```

- Exports only T, but not its constructors

```
module M (T(C,D,...)) where
data T = ...
```

- Exports T and its constructors C, D, ...

```
module M (T(..)) where
data T = ...
```

- Exports T and all of its constructors

Exporting data types

```
module M (T) where
data T = ...
```

- Exports only T, but not its constructors

```
module M (T(C,D,...)) where
data T = ...
```

- Exports T and its constructors C, D, . . .

```
module M (T(..)) where
data T = ...
```

- Exports T and all of its constructors

Not permitted: `module M (T,C,D) where`

Exporting modules

By default, modules do not export names from imported modules

Exporting modules

By default, modules do not export names from imported modules

```
module B where
import A
...
```

Exporting modules

By default, modules do not export names from imported modules

```
module B where                module A where
import A                      f = ...
...                           ...
```

⟹ B does not export f

Unless the names are mentioned in the export list

Exporting modules

By default, modules do not export names from imported modules

```
module B where              module A where
import A                    f = ...
...                         ...
```

⟹ B does not export f

Unless the names are mentioned in the export list

```
module B (f) where
import A
...
```

Exporting modules

By default, modules do not export names from imported modules

```
module B where              module A where
import A                    f = ...
...                         ...
```

⟹ B does not export f

Unless the names are mentioned in the export list

```
module B (f) where
import A
...
```

Or the whole module is exported

Exporting modules

By default, modules do not export names from imported modules

```
module B where              module A where
import A                    f = ...
...                         ...
```

⟹ B does not export f

Unless the names are mentioned in the export list

```
module B (f) where
import A
...
```

Or the whole module is exported

```
module B (module A) where
import A
...
```

import

import

By default, everything that is exported is imported

```
module B where
import A
...
```

import

By default, everything that is exported is imported

```
module B where                module A where
import A                      f = ...
...                           g = ...
```

⟹ B imports `f` and `g`

Unless an import list is specified

import

By default, everything that is exported is imported

```
module B where                module A where
import A                      f = ...
...                           g = ...
```

⟹ B imports `f` and `g`

Unless an import list is specified

```
module B where
import A (f)
...
```

⟹ B imports only `f`

Or specific names are hidden

import

By default, everything that is exported is imported

```
module B where                module A where
import A                      f = ...
...                           g = ...
```

⟹ B imports `f` and `g`

Unless an import list is specified

```
module B where
import A (f)
...
```

⟹ B imports only `f`

Or specific names are hidden

```
module B where
import A hiding (g)
...
```

qualified

qualified

```
import A
import B
import C
... f ...
```

Where does f come from??

qualified

```
import A
import B
import C
... f ...
```

Where does f come from??

Clearer: *qualified names*

```
... A.f ...
```

Renaming modules

Renaming modules

Renaming modules

```
import TotallyAwesomeModule

... TotallyAwesomeModule.f ...
```

Painful

Renaming modules

```
import TotallyAwesomeModule

... TotallyAwesomeModule.f ...
```

Painful

More readable:

```
import qualified TotallyAwesomeModule as TAM
```

Renaming modules

```
import TotallyAwesomeModule

... TotallyAwesomeModule.f ...
```

Painful

More readable:

```
import qualified TotallyAwesomeModule as TAM

... TAM.f ...
```

For the full description of the module system
see the Haskell report

qualified

```
import A
import B
import C
... f ...
```

Where does `f` come from??

Clearer: *qualified names*

```
... A.f ...
```

Can be enforced:

```
import qualified A
```

⟹ must always write `A.f`

Renaming modules

```
import TotallyAwesomeModule

... TotallyAwesomeModule.f ...
```

Painful

More readable:

```
import qualified TotallyAwesomeModule as TAM

... TAM.f ...
```

For the full description of the module system
see the Haskell report

10.2 Abstract Data Types