

Class 9: Input-Output

March 26

side effects in a pure language

```
main :: IO ()  
main = putStrLn "Hello, world!"
```

printing example

“values of type `IO a` are **descriptions** of effectful computations”

the IO type constructor

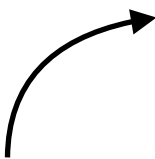
:: Recipe Cake

there is no cake inside a recipe for cake

```
:: IO String
```

there is no String inside IO String

`main :: IO ()`
`main = putStrLn "Hello, world!"`



unit type

printing example

useful IO functions


```
putStr :: String -> IO ()
```

```
putStrLn :: String -> IO ()
```

output functions

```
> print 3  
3
```

```
> print True  
True
```

output functions

```
print :: Show a => a -> IO ()
```

output functions

```
getLine :: IO String
```

input functions

```
readLn :: Read a => IO a
```

input functions

combining IO computations

$$(>>) :: IO\ a \rightarrow IO\ b \rightarrow IO\ b$$

`ex1 :: IO ()`

`ex1 = putStr "Hello, " >> putStr "world!"`

`ex2 :: IO Int`

`ex2 = putStr "Enter a number: " >> readLn`

“and then”

```
(>>=) :: IO a -> (a -> IO b) -> IO b
```

```
ex3 :: IO ()
```

```
ex3 = putStr "Enter a number: "
```

```
    >> readLn
```

```
    >>= (\n -> print (n + 1))
```

“bind”


```
ex1 :: IO ()
```

```
ex1 = putStr "Hello, " >> putStr "world!"
```

```
ex1' :: IO ()
```

```
ex1' = do
```

```
    putStr "Hello, "
```

```
    putStr "world!"
```

“do” notation

```
ex2 :: IO Int
```

```
ex2 = putStr "Enter a number: " >> readLn
```

```
ex2' :: IO Int
```

```
ex2' = do
```

```
    putStr "Enter a number: "
```

```
    readLn
```

“do” notation

```
ex3 :: IO ()  
ex3 = putStr "Enter a number: "  
      >> readLn  
      >>= (\n -> print (n + 1))
```

```
ex3' :: IO ()  
ex3' = do  
  putStr "Enter a number: "  
  n <- readLn  
  print (n + 1)
```

“do” notation

(exercise: input two numbers)

```
return :: a -> IO a
```

```
ex4 :: IO Int
```

```
ex4 = getLine
```

```
    >>= (\input ->
```

```
        return (length input))
```

```
ex4' :: IO Int
```

```
ex4' = do
```

```
    input <- getLine
```

```
    return (length input)
```

“do” notation

```
ex5 :: IO ()
ex5 = do
    putStr "Give me five: "
    n <- readLn
    if n == 5
        then putStrLn "Hooray!"
    else do
        putStrLn "Try again."
    ex5
```

“do” notation

(exercise: guess a number)

```
printList :: Show a => [a] -> IO ()  
printList [] = return ()  
printList (x : xs) = do  
    print x  
    printList xs
```



```
map :: (a -> b) -> [a] -> [b]
```

```
printList :: Show a => [a] -> ??  
printList xs = map print xs
```

```
map :: (a -> b) -> [a] -> [b]
```

```
printList :: Show a => [a] -> [IO ()]  
printList xs = map print xs
```

```
mapM :: (a -> IO b) -> [a] -> IO [b]
```

```
printList :: Show a => [a] -> ??  
printList xs = mapM print xs
```

```
mapM :: (a -> IO b) -> [a] -> IO [b]
```

```
printList :: Show a => [a] -> IO [()]  
printList xs = mapM print xs
```

```
mapM_ :: (a -> IO b) -> [a] -> IO ()
```

```
printList :: Show a => [a] -> IO ()  
printList xs = mapM_ print xs
```