# Side Effects

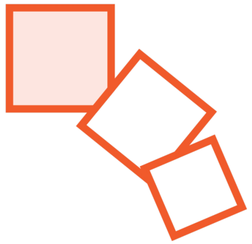
**Zachary Bennett**
Software Engineer

@z_bennett_ zachbennettcodes.com

# Side effect

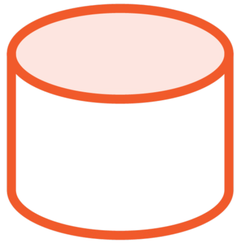
**A side effect occurs when a function interacts with data outside of itself in some way**

# Examples



**Reading or modifying a variable that exists outside of the function**

**File input/output**



**Interacting with a database**

Side effects are practical!

# Two Types of Functions

## Pure

**A pure function is a function that produces no side effects**

## Impure

**An impure function has side effects and should return “nil”**

Endeavor to create pure functions
as much as possible!

```
; Simplest example of a side effect – writing to the console

(println “If you are seeing this message, a side effect occurred!”)


; The function call above returns “nil”
```

# Side Effects and “nil”

**It is common practice for functions that perform side effects in Clojure to return “nil”. Writing to the console is the simplest example of a side effect.**

# File IO

# Very useful side effect

# Reading/Writing

## Line-by-line/Chunks

**Read/write files line-by-line – this is memory efficient**

## Strings

**Read/write files using whole strings**

```
;; Reading/Writing Strings to Files

; spit

(spit “my-file.txt” “Hello world!”)


(spit “my-file.txt” “Hi!” :append true)



; slurp

(slurp “my-file.txt”)


(slurp “my-file.txt” :encoding “UTF-8”)
```

- Spit lets you write a string to a file
- If the file already exists, you can append a string to a file like this
- Slurp lets you read the entire contents of a file as a string
- You can also specify an optional encoding

```
;; Reading/Writing to Files Efficiently


; Write to a file one line/chunk at a time
(with-open [writer (java.io/writer “file.txt”)]
  (doseq [line my-lines]
    (.write writer line)
    (.newLine writer)))


; Read a file line by line
(with-open [reader (java.io/reader “file.txt”)]
  (doseq [line (line-seq reader)]
    (println line)))
```

- with-open is a nicety which ensures that the Java writer/reader that we use is closed implicitly upon completion

- You can use the built-in Java IO Writer class to write a sequence of strings one by one to a file

- You can use the built-in Java IO Reader class alongside the “line-seq” function to read content from a file line-by-line.

# Demo

**File IO**

- Write using spit and the Java IO Writer
- Read using slurp and the Java IO Reader

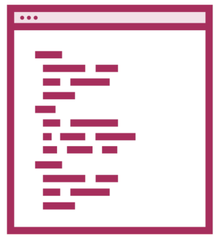
# Database Access

# Database driver

**A database driver allows you to easily connect to a specific kind of database**

# Database Driver

**Code/Program**



**Driver**

**Database**

# JDBC – Java Database Connectivity

```
(def connection-info {
  :subprotocol ”postgresql"
  :subname "//127.0.0.1:5432/coffee"
  :user "root"
  :password ”password” })
```

# JDBC Database Connection

**This is an example connection hash-map which is passed to the JDBC driver in order to create a new database connection.**

```
;; Database Access

(ns test.core
  (:require [clojure.java.jdbc :as sql]))
```

- Import the JDBC driver library

```
(println (sql/query connection-info
    ["SELECT * FROM coffee"]
     :row-fn :sequence))
```

- Connect to our database using the connection-info hash-map then fire off a SQL query

- The :sequence keyword when used in conjunction with the :row-fn keyword will return all of the rows of the return query as a sequence.

# Demo

**Database Access**

- Write data to a PostgreSQL database
- Query data from a PostgreSQL database