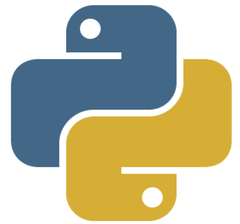
Functional Programming



Hi, I'm Alastair!

github.com/alastairparagas

Strict functional programming in **Haskell**

Relaxed functional programming in

Javascript, Python, PHP and Scala

Requirements

NodeJS 6.12.3 or later

Python 3.6 or later

Why should you care?

Makes **unit testing** and **debugging** easier

Trivial to scale computation currently done in one thread, to multiple threads, multiple processes and even, multiple servers

Easier to conceptualize and understand with declarative codebases and less lines of code

Compiler/interpreter takes the **best course of action** when trying to determine the **performance** of your code.

FAD Acronym

Functions for everything

Avoid/Isolate Side Effects

Data is immutable

Functions for Everything

Functions as inputs/parameters

Don’t be afraid to use functions as return values

Complex functions built with simpler functions

```
/**
* @param {int} initValue
* @param {Function} incrementEventListener - Function that fires
*   everytime the counter variable is incremented. Passed the
*   new counter value
* @returns {Function} increments the closured counter variable
*/
function intIncrementer(initValue, incrementEventListener) {

  var counter = initValue;

  return function increment() {
    counter++;
    incrementEventListener(counter);
  }

}

const zeroIntIncrementer = intIncrementer(0, function (counterVal) {
  console.log("Counter value incremented - new value: " + counterVal);
});

zeroIntIncrementer();
zeroIntIncrementer();
zeroIntIncrementer();
```

```
"""
@param {int} initValue
@param {Function} incrementEventListener - Function that fires
  everytime the counter variable is incremented. Passed the
  new counter value
@returns {Function} increments the closured counter variable
"""
def intIncrementer(initValue, incrementEventListener):

  counter = initValue

  def increment():
    nonlocal counter
    counter = counter + 1
    incrementEventListener(counter)

  return increment

zeroIntIncrementer = intIncrementer(
  0,
  lambda counter: print("Counter value incremented - new value: {counterVal}".format(
    counterVal=counter
  ))
)

zeroIntIncrementer()
zeroIntIncrementer()
zeroIntIncrementer()
```

Avoid/Isolate Side Effects

Storing something in the **database**

Displaying **graphics**

Printing out **text**

Modifying a **global variable**

```
var pg = require('pg');

/**
* Interfaces with the database, allowing us to
*   execute SQL-based commands
* @param {string} sqlQuery - Prepared Query to execute
* @param {[]} queryValues - Corresponding data to execute query with
* @param {Promise} resolved on success, reject on failure
*/
function dbQuery(sqlQuery, queryValues) {

    return new Promise(function (resolve, reject) {
        pg.connect(process.env.DATABASE_URI, function (err, client, done) {
            // Handle Errors
            if (err) {
                if (client) {
                    done(client);
                } else {
                    done();
                }
                reject(err);
                return;
            }

            // Query Object
            var values = queryValues || [],
                query = client.query(sqlQuery, values),
                resultList = [],

                rowCallback = function (row) {
                    resultList.push(row);
                },
                errorCallback = function (error) {
                    done();
                    reject(error);
                };

            query.on('row', rowCallback);
            query.on('error', errorCallback);

            query.on('end', function () {
                query.removeListener('row', rowCallback);
                query.removeListener('error', errorCallback);
                done();
                resolve(resultList);
            });
        });
    });
}

module.exports = exports = dbQuery;
```

```python
import psycopg2
import os

'''
Executes the provided SQL query
@param {string} sql_query
@param {list} parameters
@returns {list[tuple]}
'''
def db_query(sql_query, parameters=None):
  if not isinstance(sql_query, basestring):
    raise TypeError("sql_query must be a string")
  if parameters is not None and not isinstance(parameters, list):
    raise TypeError("parameters must be a list")

  db = psycopg2.connect(
    database=os.getenv("POSTGRES_DB"),
    user=os.getenv("POSTGRES_USER"),
    password=os.getenv("POSTGRES_PASSWORD"),
    host=os.getenv("POSTGRES_HOST")
  )

  db_cursor = db.cursor()
  if parameters is None:
    db_cursor.execute(sql_query)
  else:
    db_cursor.execute(sql_query, parameters)
  db.commit()

  results = db_cursor.fetchall()

  db_cursor.close()
  db.close()
  return results
```

Data is Immutable

Don't mutate **variables**, **data structures**

or **input parameters** - depend on results of

functions, generate new data structures and treat

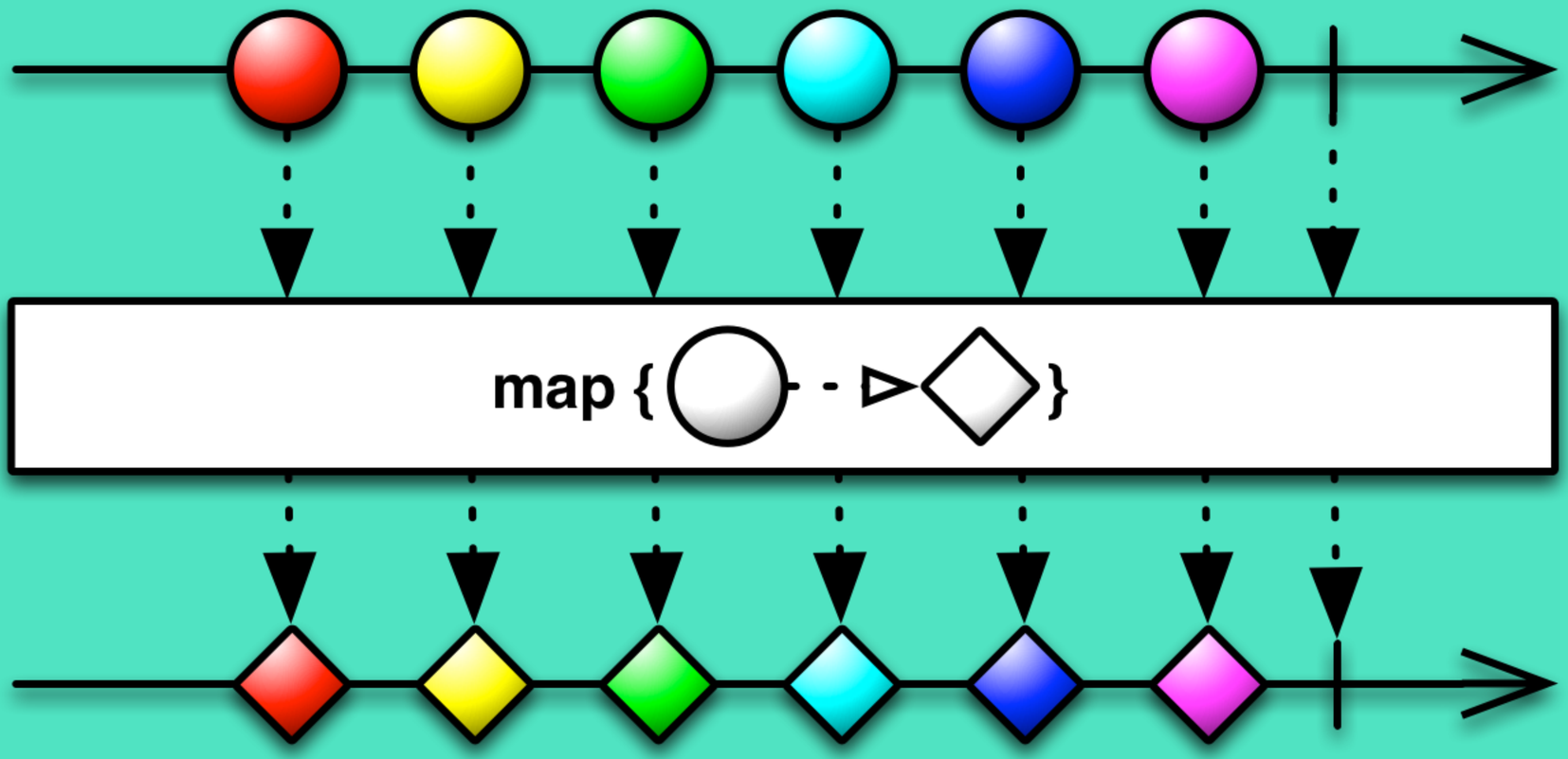
input parameters as constants (when possible)

```
function mutatesParams(mutableObjectParam) {
  mutableObjectParam.newAddedProperty = "someValue";
  return mutableObjectParam;
}

var someObject = {
  key: "value"
};

var returnValue1 = mutatesParams(someObject);
console.log('newAddedProperty' in someObject);
console.log('newAddedProperty' in returnValue1);
console.log(someObject === returnValue1);
```

```
function doesNotMutateParams(mutableObjectParam) {
  var returnedObject = Object.assign({}, mutableObjectParam);

  returnedObject.newAddedProperty = "someValue";
  return returnedObject;
}

var someOtherObject = {
  key: "value"
};

var returnValue2 = doesNotMutateParams(someOtherObject);
console.log('newAddedProperty' in someOtherObject);
console.log('newAddedProperty' in returnValue2);
console.log(someOtherObject === returnValue2);
```

```python

def mutatesParams(mutableObjectParam):
  mutableObjectParam["newAddedProperty"] = "someValue"
  return mutableObjectParam

someObject = {
  "key": "value"
}

returnValue1 = mutatesParams(someObject)
print("newAddedProperty" in someObject.keys())
print("newAddedProperty" in someObject.keys())
print(someObject == returnValue1)



import copy

def doesNotMutateParams(mutableObjectParam):
  returnedObject = copy.deepcopy(mutableObjectParam)

  returnedObject["newAddedProperty"] = "someValue"
  return returnedObject

someOtherObject = {
  "key": "value"
}

returnValue2 = doesNotMutateParams(someOtherObject);
print("newAddedProperty" in someOtherObject.keys())
print("newAddedProperty" in returnValue2.keys())
print(someOtherObject == returnValue2)

```

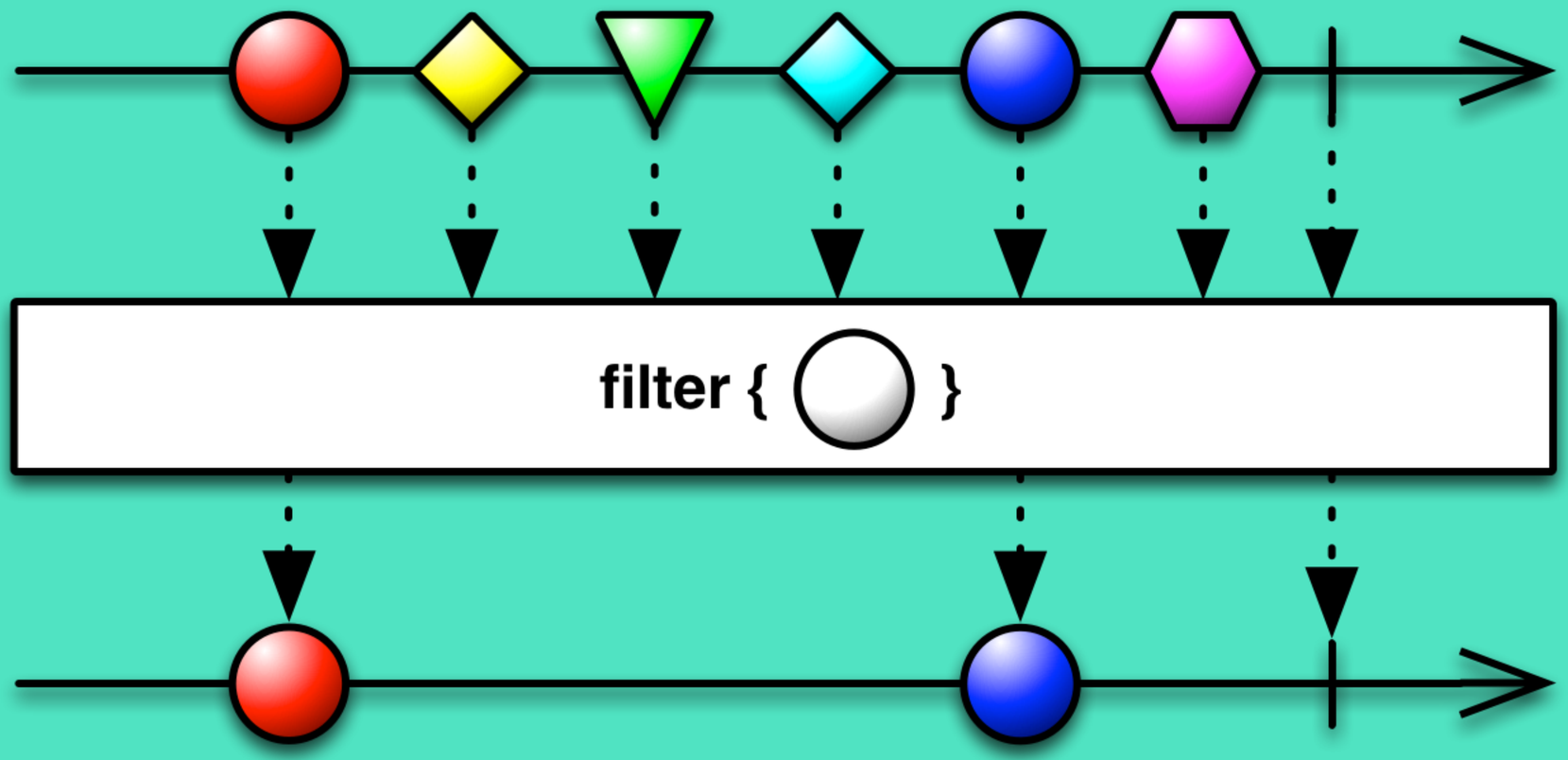
Most important functional operations


map {
}

```
var results1 = [1, 2, 3, 4, 5, 6].map(function (x) {
  return x++;
});

var results2 = ["ca", "fa", "ma", "ra"].map(function (x) {
  return x + "t";
});
```

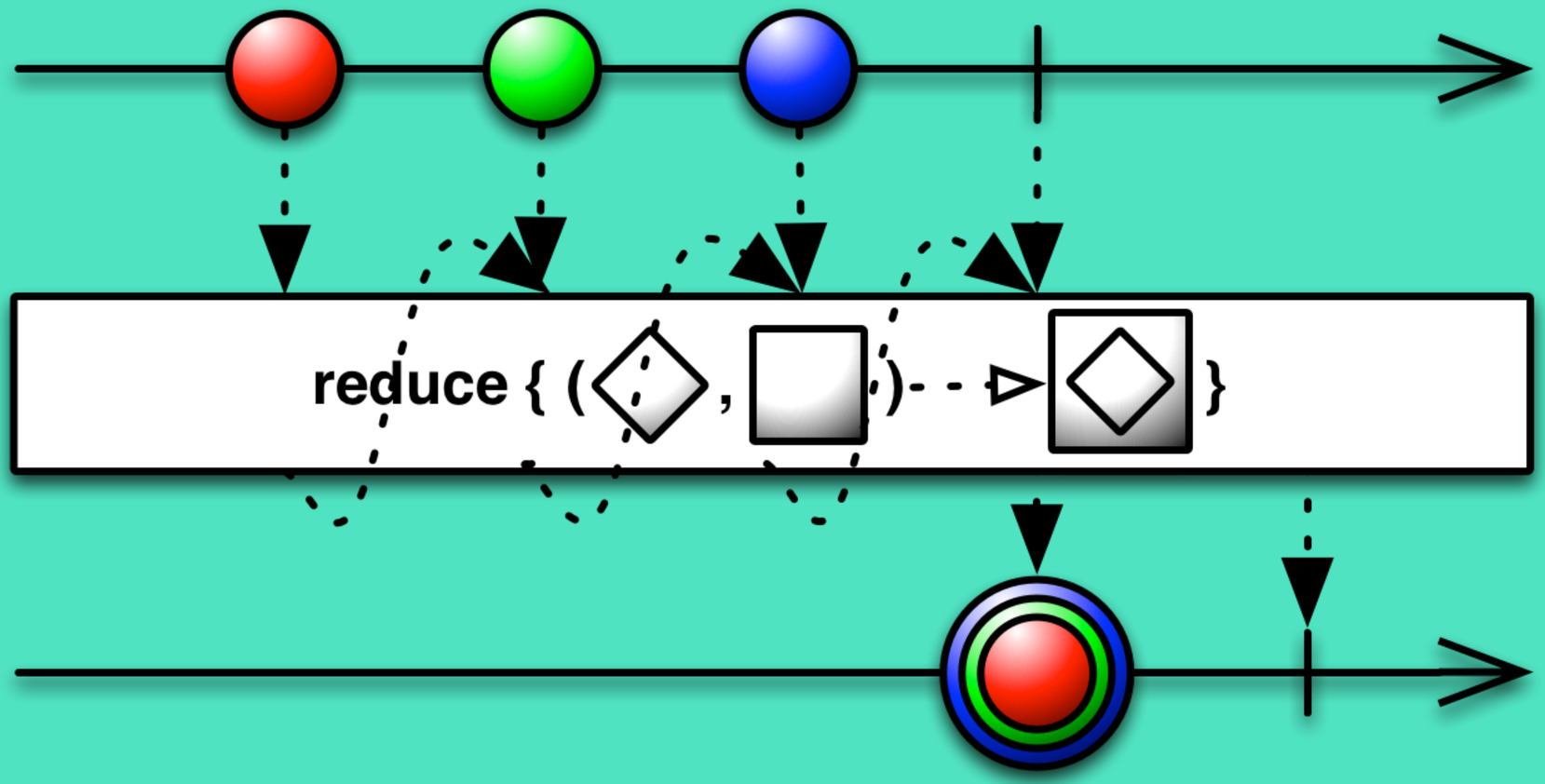
```
var results3 = [true, false].map(function (x) {
  return !x;
});

console.log(results1);
console.log(results2);
console.log(results3);
```

```
results1 = map(lambda x: x + 1, [1, 2, 3, 4, 5, 6]);

results2 = map(lambda x: x + "t", ["ca", "fa", "ma", "ra"]);

results3 = map(lambda x: not x, [True, False]);

print(list(results1))
print(list(results2))
print(list(results3))
```


filter { }

```
var results1 = ["empire", "test", "cat", "ghost"].filter(function (x) {
  return x.length > 3;
});

var results2 = [1, 1, 5, 7, 8, 9, 1, 5].filter(function (x) {
  return x > 3;
});

console.log(results1);
console.log(results2);
```

```
results1 = filter(lambda x: len(x) > 3, ["empire", "test", "cat", "ghost"])

results2 = filter(lambda x: x > 3, [1, 1, 5, 7, 8, 9, 1, 5])

print(list(results1));
print(list(results2));
```


reduce { (,) - - ▷ }

```
var results1 = [1, 1, 3, 4, 5].reduce(function (current, accumulated) {
  return current * accumulated;
});

var results2 = ["ca", "fa", "ma", "ra"].reduce(function (current, accumulated) {
  return current + accumulated;
});

console.log(results1);
console.log(results2);
```

```
from functools import reduce

def reducer1(current, accumulated):
  return current * accumulated
results1 = reduce(reducer1, [1, 1, 3, 4, 5]);

def reducer2(current, accumulated):
  return current + accumulated
results2 = reduce(reducer2, ["ca", "fa", "ma", "ra"]);

print(results1);
print(results2);
```

Cool Real-World Example - Multi-threaded HTTP Requests

Immutable Data Structures

Python

github.com/tobgu/pyrsistent

Javascript

facebook.github.io/immutable-js/

github.com/swannodette/mori

Further Learning - Resources

Picking up **Haskell**

learnyouahaskell.com

cis.upenn.edu/~cis194/spring13/

Functional **Python**

docs.python.org/3/library/functional.html

Functional **Javascript**

https://github.com/lukehoban/es6features

Further Learning - Codebases

Keyword Recommendation/Search Engine (Scala)

github.com/alastairparagas/ZenodoAddon

Database Loading Script (Python)

github.com/alastairparagas/ZenodoFilescript

Linear Regression, Twitter APIs (Haskell)

github.com/alastairparagas/haskell_exercises

Natural Language Processing Microservice (Java)

github.com/termmerge/nlpcore