Πανεπιστήμιο Μακεδονίας

Τμήμα Εφαρμοσμένης Πληροφορικής

Συναρτησιακός Προγραμματισμός στην JavaScript

Σαφαρίδης Φέλιξ (mai16020)

Θεσσαλονίκη 2019

Συναρτησιακός Προγραμματισμός

Προγραμματιστικό Μοντέλο
(Programming Paradigm)

1. Αγνές Συναρτήσεις (pure functions)
   f(x) = x + 1, f(3) = 4

2. Χωρίς παρενέργειες (no side-effects)
   x = 1;
   f(y) = y + x, f(3) = 4, x = 2; f(3) = 5

Αναφορική Ακεραιότητα (Referential Transparency)

Λογισμός Λάμδα

Alonzo Church, 1930

Συναρτησιακές αφαιρέσεις (abstractions)

Συναρτησιακές εφαρμογές (applications)

---

expression: <name> | <abstraction> | <application>

name -> έκφραση-μεταβλητή για την αναπαράσταση μίας τιμής

abstraction -> έκφραση της μορφής (λname.expression)

application -> έκφραση της μορφής (expression expression)

Παραδείγματα εκφράσεων λάμδα

def adder = λx.λy.(x + y)

adder 2 = λx.λy.(x + y) 2 = λy.(2 + y)

adder 2 3 = λx.λy.(x + y) 2 3 = λy.(2 + y) 3 =
= 2 + 3 = 5

---

def applier = λs.λa.λb.(s a b)

applier adder = λs.λa.λb.(s a b) =
= λa.λb.(λx.λy.(x + y) a b)

applier adder 3 4 = λa.λb.(λx.λy.(x + y) a b) 3 4 =
= λx.λy.(x + y) 3 4 = 7

JavaScript

```
def adder = λx.λy.(x + y)
```

```
var adder = function (x) {
    return function (y) {
        return x + y;
    };
};
```

```
def applier = λs.λa.λb.(s a b)
```

```
var applier = function (f) {
    return function (x) {
        return function (y) {
            return f(x)(y);
        };
    };
};
```

JavaScript – ES6 and beyond

```
def adder = λx.λy.(x + y)
```

```
const adder = x => y => x + y;
```

```
def applier = λs.λa.λb.(s a b)
```

```
const applier = f => x => y => f(x)(y);
```

```
const adder = x => y => x + y;
const applier = f => x => y => f(x)(y);
```

λ

JS

- adder 2 = λx.λy.(x + y) 2 = λy.(2 + y)

```
adder(2) // y => 2 + y
```

- adder 2 3 = λx.λy.(x + y) 2 3 = λy.(2 + y) 3 =
= 2 + 3 = 5

```
adder(2)(3) // 5
```

- applier adder = λs.λa.λb.(s a b) =
= λa.λb.(λx.λy.(x + y) a b)

```
applier(adder) // x => y => adder(x)(y)
               // adder
```

- applier adder 3 4 = λa.λb.(λx.λy.(x + y) a b) 3 4 =
= λx.λy.(x + y) 3 4 = 7

```
applier(adder)(3)(4) // 7
```

Δηλωτικός Προγραμματισμός (Declarative Programming)

```
const w = [1, 2, 3, 4, 5, 6];

const e = map(x => x + 1, w);

e // [2, 3, 4, 5, 6, 7]
w // [1, 2, 3, 4, 5, 6]
```

Προστακτικός Προγραμματισμός (Imperative Programming)

```
const w = [1, 2, 3, 4, 5, 6];

const f = x => x + 1;

for (let i = 0; i < w.length; i++) {
    w[i] = f(w[i]);
};

w // [2, 3, 4, 5, 6, 7]
```

Δηλωτικός Προγραμματισμός (Declarative Programming)

```
const map = (f, [first, ...rest]) => first === undefined
          ?   <?>
          : f(first) ⨁ map(f, rest)


const concat = ⨁ = (arr1, arr2) => [...arr1, ...arr2];


const map = (f, [first, ...rest]) => first === undefined
          ? <?>
          : [f(first)] ⨁ map(f, rest)


const map = (f, [first, ...rest]) => first === undefined
          ? []
          : [f(first)] ⨁ map(f, rest)


const map = (f, [first, ...rest]) => first === undefined
          ? []
          : concat([f(first)], map(f, rest))
```

Προστακτικός Προγραμματισμός (Imperative Programming)

```
const w = [1, 2, 3, 4, 5, 6];

const f = x => x + 1;

for (let i = 0; i < w.length; i++) {
    w[i] = f(w[i]);
};

w // [2, 3, 4, 5, 6, 7]
```

Δηλωτικός Προγραμματισμός (Declarative Programming)

```
const map = (f, [first, ...rest]) => first === undefined
          ? []
          : concat([f(first)], map(f, rest))
```

```
const w = [1, 2, 3, 4, 5, 6];

const e = map(x => x + 1, w);
```

```
// concat([f(1)], concat([f(2)], concat([f(3)], concat([f(4)], concat([f(5)], concat([f(6)], [])))))
```

Προστακτικός Προγραμματισμός (Imperative Programming)

```
const w = [1, 2, 3, 4, 5, 6];

const f = x => x + 1;

for (let i = 0; i < w.length; i++) {
    w[i] = f(w[i]);
};

w // [2, 3, 4, 5, 6, 7]
```

Δηλωτικός Προγραμματισμός (Declarative Programming)

```
const map = (f, [first, ...rest]) => first === undefined
        ? []
        : concat([f(first)], map(f, rest))

const foldr = (f, z, [first, ...rest]) => first === undefined
        ? z
        : f(first, foldr(f, z, rest));



const map = (f, arr) => foldr((x, y) => concat([f(x)], y), [], arr);


const w = [1, 2, 3, 4, 5, 6];

const e = map(x => x + 1, w);

// concat([f(1)], concat([f(2)], concat([f(3)], concat([f(4)], concat([f(5)], concat([f(6)], [])))))
// [2, 3, 4, 5, 6, 7]
```

Προστακτικός Προγραμματισμός (Imperative Programming)

```
const w = [1, 2, 3, 4, 5, 6];

const f = x => x + 1;

for (let i = 0; i < w.length; i++) {
    w[i] = f(w[i]);
};

w // [2, 3, 4, 5, 6, 7]
```

Δηλωτικός Προγραμματισμός (Declarative Programming)

```
const foldl = (f, z, [first, ...rest]) => first === undefined
        ? z
        : foldl(f, f(z, first), rest);


const map = (f, arr) => foldl((x, y) => concat(x, [f(y)]), [], arr);
```

```
const w = [1, 2, 3, 4, 5, 6];

const e = map(x => x + 1, w);

// concat(concat(concat(concat(concat(concat([], [f(1)]), [f(2)]), [f(3)]), [f(4)]), [f(5)]), [f(6)])
// [2, 3, 4, 5, 6, 7]
```

Προστακτικός Προγραμματισμός (Imperative Programming)

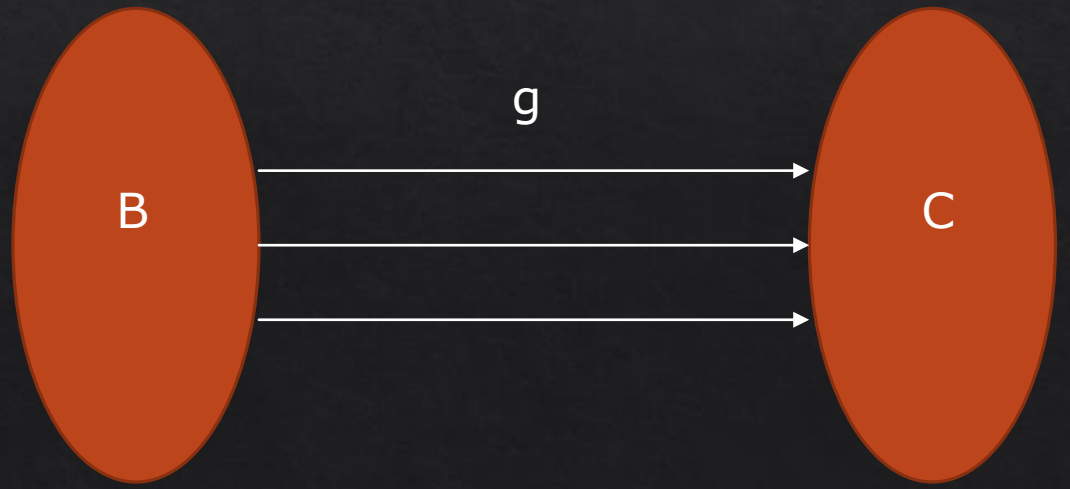
```
const w = [1, 2, 3, 4, 5, 6];

const f = x => x + 1;

for (let i = 0; i < w.length; i++) {
    w[i] = f(w[i]);
};

w // [2, 3, 4, 5, 6, 7]
```

Τύποι (Types)

```
// greet :: (String, String) -> String
const greet = (fname, lname) => `Hello mr. ${lname} ${fname}`;

// greet :: String -> (String -> String) | String -> String -> String
const greet = fname => lane => `Hello mr. ${lname} ${fname}`

        // Circle :: (Number, Number, Number) -> Circle
        const Circle = (x, y, r) => ({
            x,
            y,
            r
        });

        // area :: Circle -> Number
        const area = c => Math.PI * c.r * c.r;

        area(Circle(0, 0, 2));
        // 12.566370614359172
```

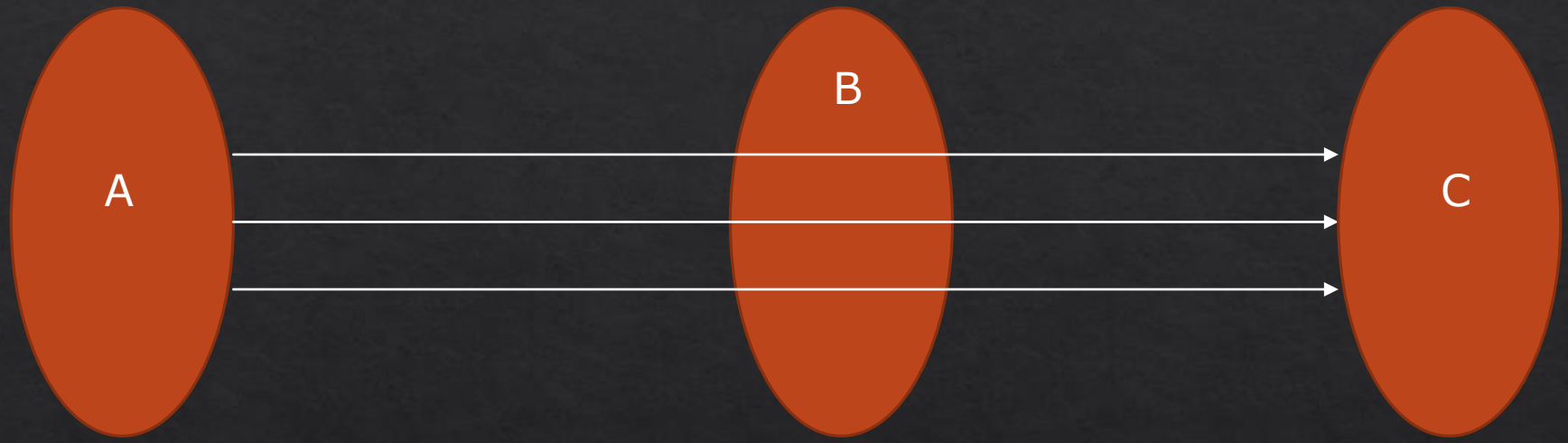
Σύνθεση Συναρτήσεων (Composition)

f: A -> B



g: B -> C

Σύνθεση Συναρτήσεων (Composition)



$g \circ f: A \to C$ με $(g \circ f)(x) = g(f(x))$

Σύνθεση Συναρτήσεων (Composition)

$g \circ f: A \to C$ με $(g \circ f)(x) = g(f(x))$

def compose = λg.λf.λx. (g (f x))

```
// compose :: (b -> c) -> (a -> b) -> (a -> c)
const compose = g => f => x => g(f(x));

// compose :: (b -> c, a -> b) -> (a -> c)
const compose = (g, f) => x => g(f(x));
```

Σύνθεση Συναρτήσεων (Composition)

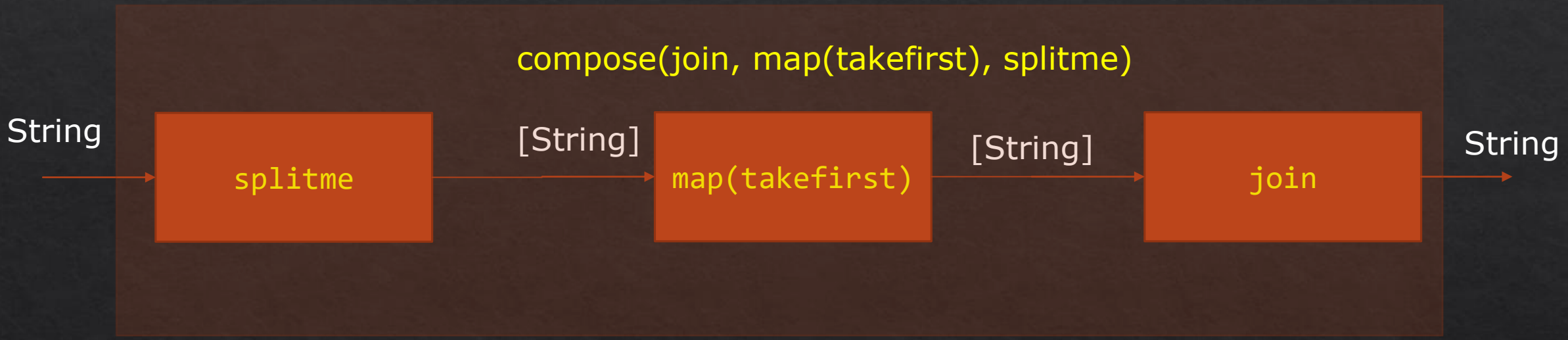
```
// compose :: (b -> c, a -> b) -> (a -> c)
const compose = (g, f) => x => g(f(x));
```

```
// splitme :: String -> [String]
const splitme = name => name.split(" ");

// map :: (String -> String) -> [String] -> [String]
const map = f => s => s.map(f);

// takefirst :: String -> String
const takefirst = s => s[0];

// join :: [String] -> String
const join = s => s.join("");

// app :: String -> String
const app = compose(join, compose(map(takefirst), splitme));

app("Felix Safaridis");
// FS
```

Σύνθεση Συναρτήσεων (Composition)

```
// splitme :: String -> [String]
const splitme = name => name.split(" ");


// map :: (String -> String) -> [String] -> [String]
const map = f => s => s.map(f);

// takefirst :: String -> String
const takefirst = s => s[0];


// join :: [String] -> String
const join = s => s.join("");
```

Σύνθεση Συναρτήσεων (Composition)




```
// compose :: (c -> d, b -> c, a -> b) -> (a -> d)
const compose = (h, g, f) => x => h(g(f(x)));
```

Currying

def cadd = λx.λy.λz.(x + y + z)

```
// add :: (Number, Number, Number) -> Number
const add = (x, y, z) = x + y + z;
add(2) // Not a Function

// cadd :: Number -> Number -> Number -> Number
const cadd = x => y => z => x + y + z;

// curry3 :: ((a, b, c) -> d) -> a -> b -> c -> d
const curry3 = f => x => y => z => f(x, y, z);

// cadd :: Number -> Number -> Number -> Number
const cadd = curry3(add);

cadd(2) // y => z => 2 + y + z
cadd(2)(3) // z => 2 + 3 + z
cadd(2)(3)(4) // 9
```

Functional Data Types

```
// Circle :: (Number, Number, Number) -> Circle
const Circle = (x, y, r) => ({
    x,
    y,
    r
});
```

```
def Circle = λx.λy.λr.λs. (s x y r)
```

```
// Circle :: Number -> Number -> Number -> Circle
const Circle = x => y => r => s => s(x)(y)(r);
```

Functional Data Types

```
// Circle :: (Number, Number, Number) -> Circle
const Circle = (x, y, r) => ({
    x,
    y,
    r
});
```

```
// area :: Circle -> Number
const area = c => Math.PI * c.r * c.r;

area(Circle(0, 0, 2));
// 12.566370614359172
```

```
// Circle :: Number -> Number -> Number -> Circle
const Circle = x => y => r => s => s(x)(y)(r);
```

```
// area :: Circle -> Number
const area = c => c(_ => _ => r => Math.PI * r * r);

area(Circle(0)(0)(2));
// 12.566370614359172
```

Functional Data Types

```haskell
data Circle = Circle Float Float Float
```

```haskell
area :: Circle -> Float
area (Circle _ _ r) = pi * r * r

area $ Circle 0 0 2
// 12.566371
```

```javascript
// Circle :: Number -> Number -> Number -> Circle
const Circle = x => y => r => s => s(x)(y)(r);
```

```javascript
// area :: Circle -> Number
const area = c => c(_ => _ => r => Math.PI * r * r);

area(Circle(0)(0)(2));
// 12.566370614359172
```

Functional Data Structures

List a : empty | cons(a, List a)

def cons = λx.λy.λs. s x y

def car = λm. m (λx.λy.x)

def cdr = λm. m (λx.λy.y)

```
const empty = null;

// cons :: a -> cons a -> cons a
const cons = x => y => s => s(x)(y);

// car :: cons a -> a
const car = m => m(x => y => x);

// cdr :: cons a -> cons a
const cdr = m => m(x => y => y);

const oneToFour = cons(1)(cons(2)(cons(3)(cons(4)(empty))));

car(oneToFour) // 1
cdr(oneToFour) // cons(2)(cons(3)(cons(4)(empty)))
car(cdr(oneToFour)) // 2
```

Functional Data Structures

List a : empty | cons(a, List a)

```
// foldr :: (b -> a -> a) -> a -> [b] -> a
const foldr = f => z => ([first, ...rest]) => first === undefined
       ? z
       : f(first)(foldr(f)(z)(rest));


// fromArray :: [b] -> cons b
const fromArray = foldr(cons)(empty);


const fiveToNine = fromArray([5, 6, 7, 8, 9]);
// cons(5)(cons(6)(cons(7)(cons(8)(cons(9)(empty)))))

const fourToNine = cons(4)(fiveToNine);
// cons(4)(cons(5)(cons(6)(cons(7)(cons(8)(cons(9)(empty))))))
```

Functional Data Structures

List a : empty | cons(a, List a)

```
// map :: (a -> b) -> cons a -> cons b
const map = f => m => m === empty
       ? empty
       : cons(f(car(m)))(map(f)(cdr(m)))
```

```
// filter :: (a -> Boolean) -> cons a -> cons a
const filter = f => m => m === empty
       ? empty
       : f(car(m))
         ? cons(car(m))(filter(f)(cdr(m)))
         : filter(f)(cdr(m))
```

Functional Data Structures

List a : empty | cons(a, List a)

```
const fiveToNine = fromArray([5, 6, 7, 8, 9]);
// cons(5)(cons(6)(cons(7)(cons(8)(cons(9)(empty)))))


const fourToNine = cons(4)(fiveToNine);
// cons(4)(cons(5)(cons(6)(cons(7)(cons(8)(cons(9)(empty))))))


map(x => x + 1)(fourToNine);
// cons(5)(cons(6)(cons(7)(cons(8)(cons(9)(cons(10)(empty))))))


filter(x => x > 5)(fourToNine);
// cons(6)(cons(7)(cons(8)(cons(9)(empty))))
```

Functional Data Structures

Stream a : delay (empty | cons(a, Stream a))

delay -> καθυστέρηση αποτίμησης (lazy evaluation) με την χρήση thunks

```
const fiveToNine = fromArray([5, 6, 7, 8, 9]);
// cons(5)(cons(6)(cons(7)(cons(8)(cons(9)(empty)))))
```

```
() => cons(5)(() => cons(6)(() => cons(7)(() => cons(8)(() => cons(9)(() => empty)))))
```

Functional Data Structures

Stream a : delay (empty | cons(a, Stream a))

delay -> καθυστέρηση αποτίμησης (lazy evaluation) με την χρήση thunks

```
// range :: (Number, Number) => cons a
const range = (n, m) => n > m
    ? empty
    : cons(n)(range(n + 1, m))

range(1, 6)
// cons(1)(cons(2)(cons(3)(cons(4)(cons(5)(cons(6)(empty))))))
```

Functional Data Structures

Stream a : delay (empty | cons(a, Stream a))

delay -> καθυστέρηση αποτίμησης (lazy evaluation) με την χρήση thunks

```
// range :: (Number, Number) => cons a
const range = (n, m) => n > m
    ? empty
    : cons(n)(range(n + 1, m))
```

```
const delay = f => (...args) => () => f(...args);
```

```
// range :: (Number, Number) => lzcons a
const range = delay(
    (n, m) => n > m
        ? empty
        : cons(n)(range(n + 1, m))
);
```

Functional Data Structures

Stream a : delay (empty | cons(a, Stream a))

delay -> καθυστέρηση αποτίμησης (lazy evaluation) με την χρήση thunks

```
// range :: (Number, Number) => lzcons a
const range = delay(
    (n, m) => n > m
        ? empty
        : cons(n)(range(n + 1, m))
);
```

```
const delay = f => (...args) => () => f(...args);
```

```
range(1, 5)
// () => cons(1)(() => cons(2)(() => cons(3)(() => cons(4)(() => cons(5)(() => empty)))))
```

Functional Data Structures

Stream a : delay (empty | cons(a, Stream a))

delay -> καθυστέρηση αποτίμησης (lazy evaluation) με την χρήση thunks

```
range(1, 5)
// () => cons(1)(() => cons(2)(() => cons(3)(() => cons(4)(() => cons(5)(() => empty)))))


// map :: (a -> b) -> lzcons a -> lzcons b
const map = delay(
    (f, m) => m() === empty
        ? empty
        : cons(f(car(m())))(map(f, cdr(m())))
);

 map(x => x + 2, range(1, 5))
 // () => cons(3)(() => cons(4)(() => cons(5)(() => cons(6)(() => cons(7)(() => empty)))))
```

Functional Data Structures

Stream a : delay (empty | cons(a, Stream a))

delay -> καθυστέρηση αποτίμησης (lazy evaluation) με την χρήση thunks

```
// map :: (a -> b) -> lzcons a -> lzcons b
const map = delay(
    (f, m) => m() === empty
        ? empty
        : cons(f(car(m())))(map(f, cdr(m())))
);
```

```
// filter :: (a -> Boolean) -> lzcons a -> lzcons a
const filter = delay(
    (f, m) => m() === empty
        ? empty
        : f(car(m()))
            ? cons(car(m()))(filter(f, cdr(m())))
            : filter(f, cdr(m()))()
);
```

```
filter(x => x % 2 === 0, map(x => -x, range(1, Infinity)))
// () => cons(-2)(() => cons(-4)(() => cons(-6)(() => cons(-8)(() => . . .)
```

Functional Data Structures

Stream a : delay (empty | cons(a, Stream a))

```
const a = filter(x => x % 2 === 0, map(x => -x, range(1, Infinity)))
// () => cons(-2)(() => cons(-4)(() => cons(-6)(() => cons(-8)(() => . . .)
```

```
const tramboline = (f, m) => {
    const init = m;
    let evaluated;
    while ((evaluated = m()) !== empty) {
        f(car(evaluated));
        m = cdr(evaluated);
    }
    return init;
};
```

```
tramboline(log, a);
// -2, -4, -6, -8, -10, ..., -20000, ...
```

Μονοειδή (Monoids)

Αλγεβρική αφηρημένη δομή

Δυαδική πράξη (binary associative operation) της μορφής α -> α -> α

Ουδέτερο στοιχείο (neutral/identity element)

Παραδείγματα

Το σύνολο ℕ υπό την πράξη της πρόσθεσης με ουδέτερο στοιχείο το 0

Το σύνολο ℕ υπό την πράξη του πολ/σμού με ουδέτερο στοιχείο το 1

Το σύνολο των STRING υπό την πράξη της συνάθροισης (concatenation) με ουδέτερο στοιχείο το ""

Arrays, Lists, Streams υπό την πράξης της συνάθροισης (concatenation) με ουδέτερα στοιχεία τα [], empty, () => empty αντίστοιχα

Μονοειδή (Monoids)

```
// mconcat :: [a] -> a
const mconcat = foldr(mappend)(mempty);
```

mappend -> δυαδική πράξη
mempty -> ουδέτερο στοιχείο

Αν S ένα σύνολο, τότε όλες οι συναρτήσεις του συνολού της μορφής S -> S σχηματίζουν μονοειδή υπό την πράξη της συναρτησιακής σύνθεσης (function composition) και με ουδέτερο στοιχείο την ταυτοτική συνάρτηση

mappend -> ∘
mempty -> f(x) = x

```
// compose :: (b -> c, a -> b) -> (a -> c)
const compose2 = (g, f) => x => g(f(x));

// id :: a -> a
const id = x => x
```

Μονοειδή (Monoids)

```
// mconcat :: [a] -> a
const mconcat = foldr(mappend)(mempty);
```

```
// mappend :: (b -> c, a -> b) -> (a -> c)
const mappend = (g, f) => x => g(f(x));
```

```
// mempty :: a -> a
const mempty = x => x;
```

```
// compose :: [(y -> z, x -> y, ..., b -> c, a -> b)] -> a -> z
const compose = mconcat;
```

```
compose([x => x + 1, x => x * 2, x => x - 10, x => x / 2])(10) // -9
```

Μονάδες (Monads)

Προσδίδουν ένα επίπεδο αφαίρεσης γύρω από ένα υπολογιστικό πλαίσιο

Αποκρύπτουν

- Διαχείριση Δεδομένων (Data Management)
- Έλεγχο Ροής (Control Flow)
- Παρενέργειες (side effects)

Παραδείγματα

- Maybe, Either, IO, State, Array, List, Streams, Writer, Task(Promise?) …etc

Μονάδες (Monads)

return/pure: a -> M a

chain/flatMap: M a -> (a -> M b) -> M b

Πίνακες

```
// pure/return :: a -> [a]
const pure = x => [x];
```

```
const mempty = []
const mappend = (arr1, arr2) => [...arr1, ...arr2];

// mconcat :: [a] -> a
const mconcat = foldr(mappend)(mempty);
```

```
// chain/flatMap :: [a] -> (a -> [b]) -> [b]
const chain = m => f => mconcat(map(m)(f));
```

Μονάδες (Monads)

```
// pure/return :: a -> [a]
const pure = x => [x];

// chain/flatMap :: [a] -> (a -> [b]) -> [b]
const chain = m => f => mconcat(map(m)(f));
```

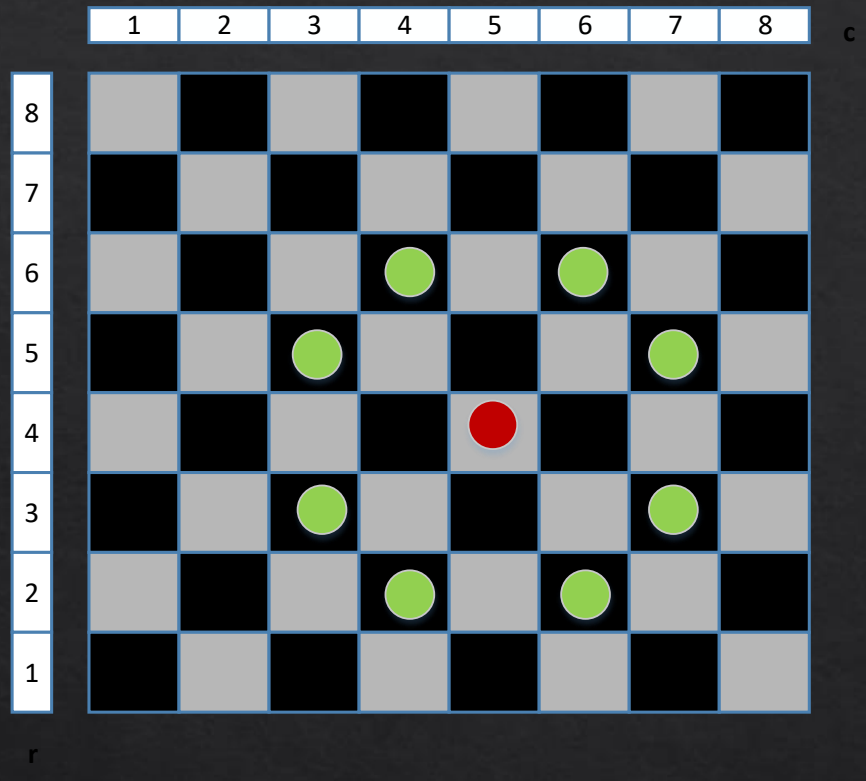
Απόκρυψη έλλειψης Ντετερμινισμού

```
const some = ["a", "b", "c"];
```

Συνδυαστική Αλληλεπίδραση

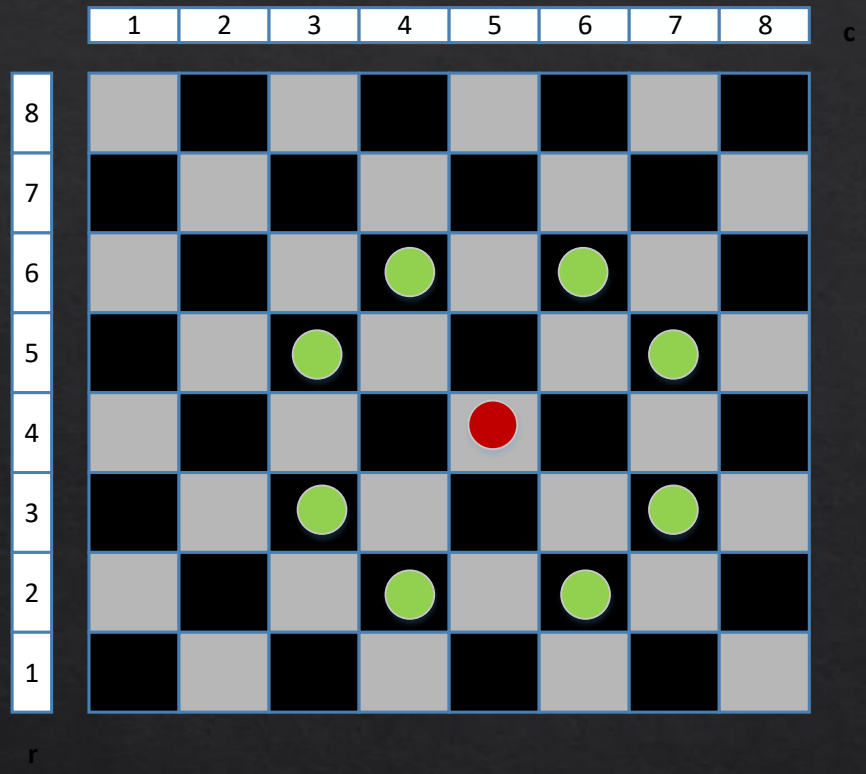
```
chain(some)(x => [`${x}x`, `${x}y`, `${x}z`])
// [ 'ax', 'ay', 'az', 'bx', 'by', 'bz', 'cx', 'cy', 'cz' ]
```

Μονάδες (Monads)



Αν μας δίνεται η αρχική θέση του ίππου να βρεθούν οι πιθανές θέσεις του μετά από n διαδοχικές κινήσεις

Μονάδες (Monads)




```
// Position :: (Number, Number) -> Position
const Position = (c, r) => ({
    col: c,
    row: r,
});

// init :: Position
const init = Position(5, 4);

// colAndRowInBound :: Position -> Boolean
const colAndRowInBound = ({col, row}) =>
    col > 0 && col < 9 && row > 0 && row < 9;

// moveHorse :: Position -> [Position]
const moveHorse = ({ col, row, prevPos }) => ([
    Position(col + 2, row + 1),
    Position(col + 2, row - 1),
    Position(col - 2, row - 1),
    Position(col - 2, row + 1),
    Position(col + 1, row - 2),
    Position(col + 1, row + 2),
    Position(col - 1, row - 2),
    Position(col - 1, row + 2)
]).filter(colAndRowInBound);
```

```
// Position :: (Number, Number -> Position
const Position = (c, r) => ({
    col: c,
    row: r,
});

// init :: Position
const init = Position(5, 4);

// colAndRowInBound :: Position -> Boolean
const colAndRowInBound = ({col, row}) =>
    col > 0 && col < 9 && row > 0 && row < 9;

// moveHorse :: Position -> [Position]
const moveHorse = ({ col, row, prevPos }) => ([
    Position(col + 2, row + 1),
    Position(col + 2, row - 1),
    Position(col - 2, row - 1),
    Position(col - 2, row + 1),
    Position(col + 1, row - 2),
    Position(col + 1, row + 2),
    Position(col - 1, row - 2),
    Position(col - 1, row + 2)
]).filter(colAndRowInBound);
```

```
const a = chain(pure(init))(moveHorse);
```

```
[ { col: 7, row: 5 },
  { col: 7, row: 3 },
  { col: 3, row: 3 },
  { col: 3, row: 5 },
  { col: 6, row: 2 },
  { col: 6, row: 6 },
  { col: 4, row: 2 },
  { col: 4, row: 6 } ]
```

1η κίνηση

```
const b = chain(a)(moveHorse);
```

```
[ { col: 5, row: 4 },
  { col: 5, row: 6 },
  { col: 8, row: 3 },
  { col: 8, row: 7 },
  { col: 6, row: 3 },
  { col: 6, row: 7 },
  { col: 5, row: 2 },
  { col: 5, row: 4 },
  { col: 8, row: 1 },
  { col: 8, row: 5 },
  { col: 6, row: 1 },
  { col: 6, row: 5 },
  { col: 5, row: 4 },
  { col: 5, row: 2 },
  { col: 1, row: 2 },
  { col: 1, row: 4 },
  { col: 4, row: 1 },
  { col: 4, row: 5 },
  { col: 2, row: 1 },
  { col: 2, row: 5 },
  { col: 5, row: 6 },
  { col: 5, row: 4 },
  { col: 1, row: 4 },
  { col: 1, row: 6 },
  { col: 4, row: 3 },
  { col: 4, row: 7 },
```

```
  { col: 2, row: 3 },
  { col: 2, row: 7 },
  { col: 8, row: 3 },
  { col: 8, row: 1 },
  { col: 4, row: 1 },
  { col: 4, row: 3 },
  { col: 7, row: 4 },
  { col: 5, row: 4 },
  { col: 8, row: 7 },
  { col: 8, row: 5 },
  { col: 4, row: 5 },
  { col: 4, row: 7 },
  { col: 7, row: 4 },
  { col: 7, row: 8 },
  { col: 5, row: 4 },
  { col: 5, row: 8 },
  { col: 6, row: 3 },
  { col: 6, row: 1 },
  { col: 2, row: 1 },
```

```
  { col: 2, row: 3 },
  { col: 5, row: 4 },
  { col: 3, row: 4 },
  { col: 6, row: 7 },
  { col: 6, row: 5 },
  { col: 2, row: 5 },
  { col: 2, row: 7 },
  { col: 5, row: 4 },
  { col: 5, row: 8 },
  { col: 3, row: 4 },
  { col: 3, row: 8 } ]
```

2η κίνηση

```
// moveHorseNTimes :: (Number, [Position]) -> [Position]
const moveHorseNTimes = (n, m) => n < 1
    ? m
    : chain(moveHorseNTimes(n - 1, m))(moveHorse);
```

ή

```
// moveHorseNTimes :: (Number, [Position]) -> [Position]
const moveHorseNTimes = (n, m) =>
    foldl(chain)(m)(Array(n).fill(moveHorse));
```

```
moveHorseNTimes(3, pure(init))
```

```
[ { col: 7, row: 5 },
  { col: 7, row: 3 },
  { col: 3, row: 3 },
  { col: 3, row: 5 },
  { col: 6, row: 2 },
    ...
```

3η κίνηση

```
const a = chain(pure(init))(moveHorse);
```

```
[ { col: 7, row: 5 },
  { col: 7, row: 3 },
  { col: 3, row: 3 },
  { col: 3, row: 5 },
  { col: 6, row: 2 },
  { col: 6, row: 6 },
  { col: 4, row: 2 },
  { col: 4, row: 6 } ]
```

1η κίνηση

```
const b = chain(a)(moveHorse);
```

```
[ { col: 5, row: 4 },
  { col: 5, row: 6 },
  { col: 8, row: 3 },
  { col: 8, row: 7 },
  { col: 6, row: 3 },
  { col: 6, row: 7 },
  { col: 5, row: 2 },
  { col: 5, row: 4 },
  { col: 8, row: 1 },
  { col: 8, row: 5 },
  { col: 6, row: 1 },
  { col: 6, row: 5 },
  { col: 5, row: 4 },
  { col: 5, row: 2 },
  { col: 1, row: 2 },
  { col: 1, row: 4 },
  { col: 4, row: 1 },
  { col: 4, row: 5 },
  { col: 2, row: 1 },
  { col: 2, row: 5 },
  { col: 5, row: 6 },
  { col: 5, row: 4 },
  { col: 1, row: 4 },
  { col: 1, row: 6 },
  { col: 4, row: 3 },
  { col: 4, row: 7 },

  { col: 2, row: 3 },
  { col: 2, row: 7 },
  { col: 8, row: 3 },
  { col: 8, row: 1 },
  { col: 4, row: 1 },
  { col: 4, row: 3 },
  { col: 7, row: 4 },
  { col: 5, row: 4 },
  { col: 8, row: 7 },
  { col: 8, row: 5 },
  { col: 4, row: 5 },
  { col: 4, row: 7 },
  { col: 7, row: 4 },
  { col: 7, row: 8 },
  { col: 5, row: 4 },
  { col: 5, row: 8 },
  { col: 6, row: 3 },
  { col: 6, row: 1 },
  { col: 2, row: 1 },

  { col: 2, row: 3 },
  { col: 5, row: 4 },
  { col: 3, row: 4 },
  { col: 6, row: 7 },
  { col: 6, row: 5 },
  { col: 2, row: 5 },
  { col: 2, row: 7 },
  { col: 5, row: 4 },
  { col: 5, row: 8 },
  { col: 3, row: 4 },
  { col: 3, row: 8 } ]
```

2η κίνηση

Συναρτησιακή υλοποίηση προγράμματος ανάκτησης και τοποθέτησης εικόνων με βάση τον όρο αναζήτησης

Ερώτηση GET στο Usplash API endpoint

Προσπέλαση απάντησης (JSON parsing)

Δημιουργία εικονοστοιχείων (create Image elements)

Τοποθέτηση εικόνων στο DOM (hook to DOM)

```
// apiURL :: String -> String -> String
const apiURL = clientId => queryTerm => `https://api.unsplash.com/search/photos?query=${queryTerm}&client_id=${clientId}`;

// pluck :: String -> a -> b
const pluck = attr => obj => obj[attr];

// createImage :: String -> ImageElement
const createImage = src => {
      const img = document.createElement("img");
      img.setAttribute("src", src);
      return img;
};

// hookTo(Impure) :: String -> ImageElement -> ()
const hookTo = id => el => document.querySelector(`#${id}`).appendChild(el);

// createImageAndHookToContainer :: String -> ()
const createImageAndHookToContainer = compose(hookTo("container"), createImage);

// parseJSON :: JSON -> [String]
const parseJSON = compose(map(compose(pluck("regular"), pluck("urls"))), pluck("results"));

// deploy :: JSON -> [()]
const deploy = compose(map(createImageAndHookToContainer), parseJSON);

// getImages(Impure) :: String -> Promise
const getImages = url => fetch(url).then(res => res.json()).then(deploy);

// app :: String -> Promise
const app = compose(getImages, apiURL(CLIENTID));
```

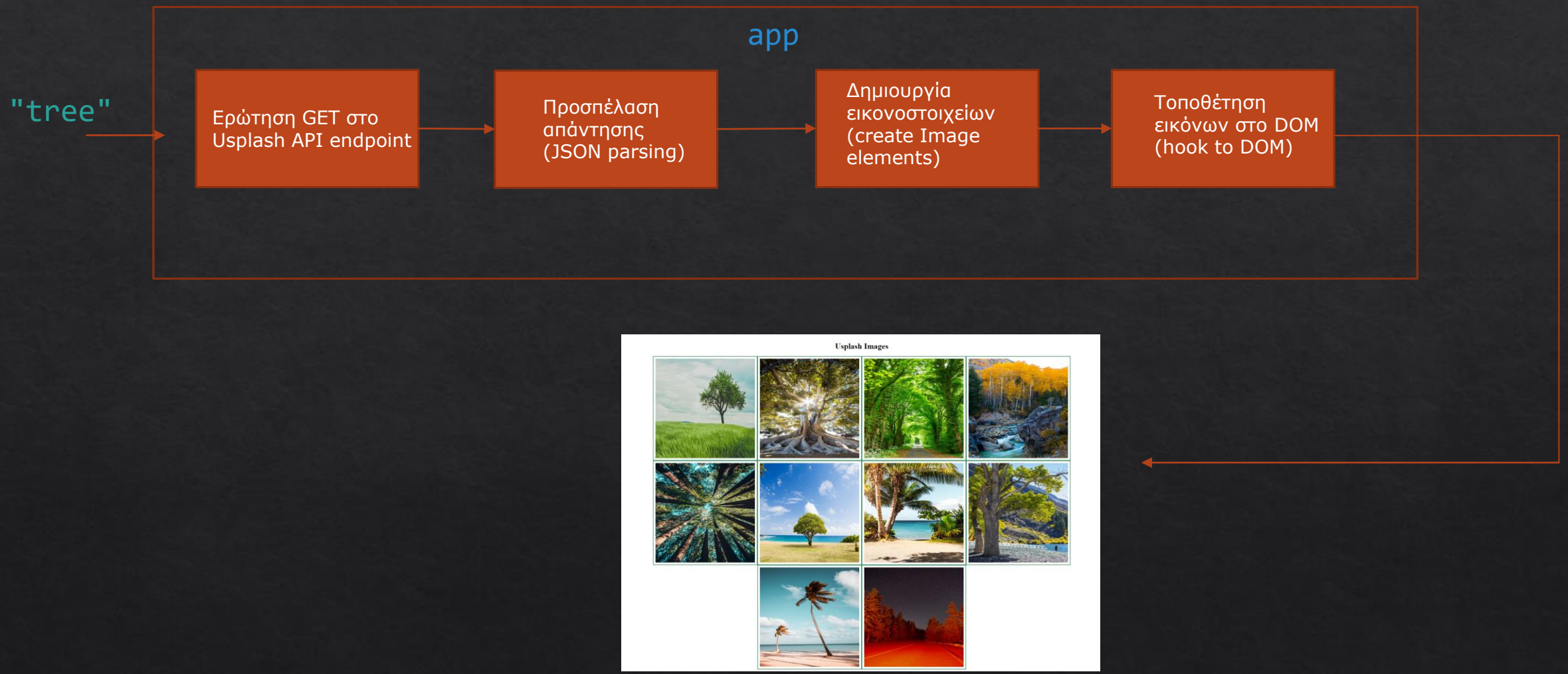
Δημιουργία εικονοστοιχείων (create Image elements)

Τοποθέτηση εικόνων στο DOM (hook to DOM)

Προσπέλαση απάντησης (JSON parsing)

Ερώτηση GET στο Usplash API endpoint

```
app("tree");
```


app
"tree"
Ερώτηση GET στο Usplash API endpoint
Προσπέλαση απάντησης (JSON parsing)
Δημιουργία εικονοστοιχείων (create Image elements)
Τοποθέτηση εικόνων στο DOM (hook to DOM)
Usplash Images

https://github.com/felichio/radiancejs

https://www.npmjs.com/package/radiancejs

Είναι ικανό το περιβάλλον της JavaScript να φιλοξενήσει κώδικα που πηγάζει από τον συναρτησιακό τρόπο σκέψης και αντίληψης;

Απουσία τύπων

Απουσία σταδίου μεταγλώττισης

Στραμμένο στην πρωτότυπη αντικειμενοστρέφεια (τροποποιήσεις και παρενέργειες, mutations and side effects)

Ο Συναρτησιακός προγραμματισμός σχηματίζει έναν νέο τρόπο σκέψης και αντίληψης (πάνω από τις γλώσσες)

Μπορεί να συνυπάρξει με το μοντέλο του αντικειμενοστρεφή προγραμματισμού (orthogonality)

?