The real power of the FUNCTION

Zdefiniuj funkcje: *range*, *map*, *reverse* i *foreach* uwzględniając poniższe ograniczenia tak, aby przedstawiony program działał poprawnie.

- Nie możesz używać tablic. Kwadratowe nawiasy *[ ]* i konstruktor *Array* są zabronione.
- Nie możesz używać obiektów. Nawiasy klamrowe *{ } użyte do tworzenia obiektu* i używanie kropki . są zabronione
- Funkcje muszą być generyczne

Nie musimy się ograniczać jedynie do 4 funkcji, możemy definiować funkcje pomocnicze.

```
let numbers = range(1, 10);
numbers = map(numbers, n => n * n);
numbers = reverse(numbers);
foreach(numbers, n => console.log(n));
// 100, 81, 64, 49, 36, 25, 16, 9, 4, 1
```

```
const range = (from, to) => {
  let result = [];
  for(let i = from; i <= to; i++) {
    result.push(i);
  }
  return result;
}
const map = (numbers, fn) => numbers.map(fn);
const reverse = numbers => numbers.reverse();
const foreach = (numbers, fn) => numbers.forEach(n =>fn(n));

let numbers = range(1, 10);
numbers = map(numbers, n => n * n);
numbers = reverse(numbers);
foreach(numbers, console.log);
// 100, 81, 64, 49, 36, 25, 16, 9, 4, 1
```

Zdefiniuj funkcje: *range*, *map*, *reverse* i foreach uwzględniając poniższe ograniczenia tak, aby program działał poprawnie.

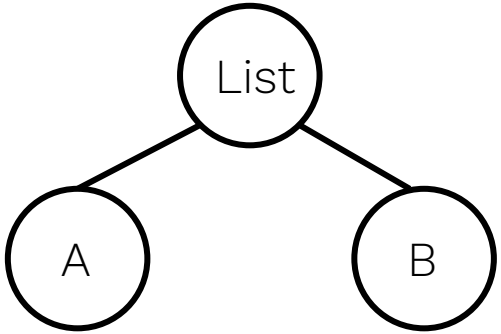
- Nie możesz używać tablic. Kwadratowe nawiasy *[ ]* i konstruktor *Array* są zabronione.
- Nie możesz używać obiektów. Nawiasy klamrowe *{ }* i używanie kropki . jest zabronione.
- Funkcje muszą być generyczne.

.

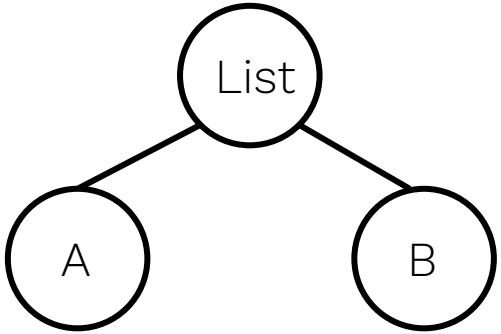
Zdefiniuj funkcje: *range*, *map*, *reverse* i foreach uwzględniając poniższe ograniczenia tak, aby program działał poprawnie.

- Nie możesz używać tablic. Kwadratowe nawiasy *[ ]* i konstruktor *Array* są zabronione.
- Nie możesz używać obiektów. Nawiasy klamrowe *{ }* i używanie kropki . jest zabronione.
- Funkcje muszą być generyczne.
- Nie możesz używać *iterator statements*, czyli:
  - *do...while*
  - *for*
  - *for...in*
  - *for...of*
  - *for await...of*
  - *while*

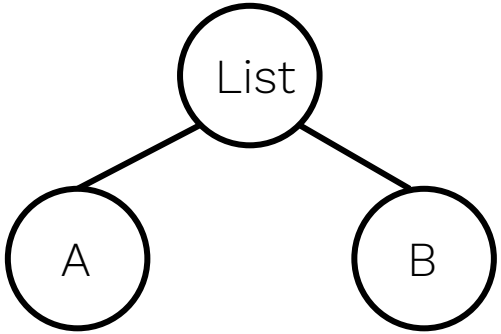
.



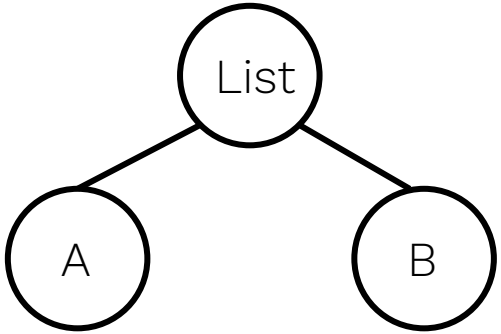

```
const myList = createList('valueA', 'valueB');
const a = first(myList);
const b = second(myList);
console.log(a, b); // valueA, valueB
```



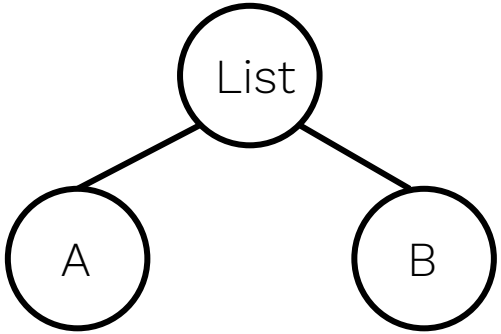

```
function createList(a, b) {
  return function(f) {
    return f(a, b);
  };
}

function first(list) {
  return list(function(a, b) {
    return a;
  });
}

function second(list) {
  return list(function(a, b) {
    return b;
  });
}

const myList = createList('valueA', 'valueB');
const a = first(myList);
const b = second(myList);
console.log(a, b); // valueA, valueB
```




```
function createList(a, b) {
  return function(f) {
    return f(a, b);
  };
}

function first(list) {
  return list(function(a, b) {
    return a;
  });
}

function second(list) {
  return list(function(a, b) {
    return b;
  });
}

const myList = createList('valueA', 'valueB');
const a = first(myList);
const b = second(myList);
console.log(a, b); // valueA, valueB
```

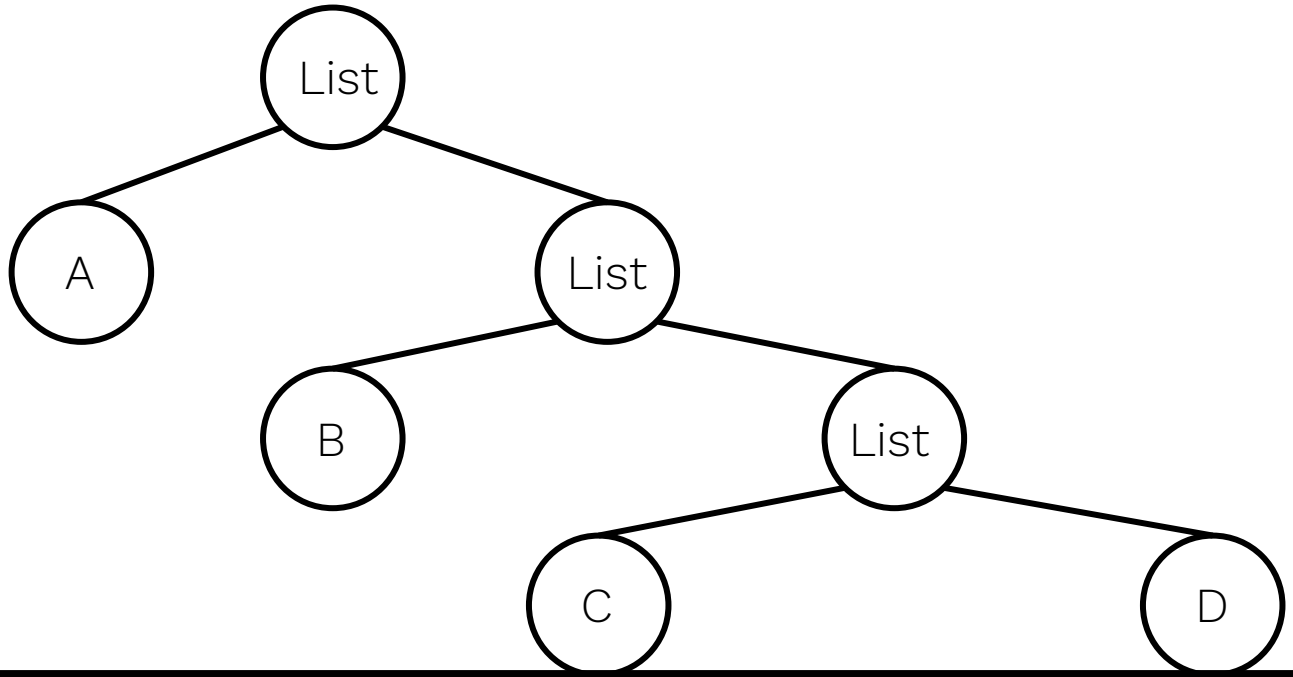


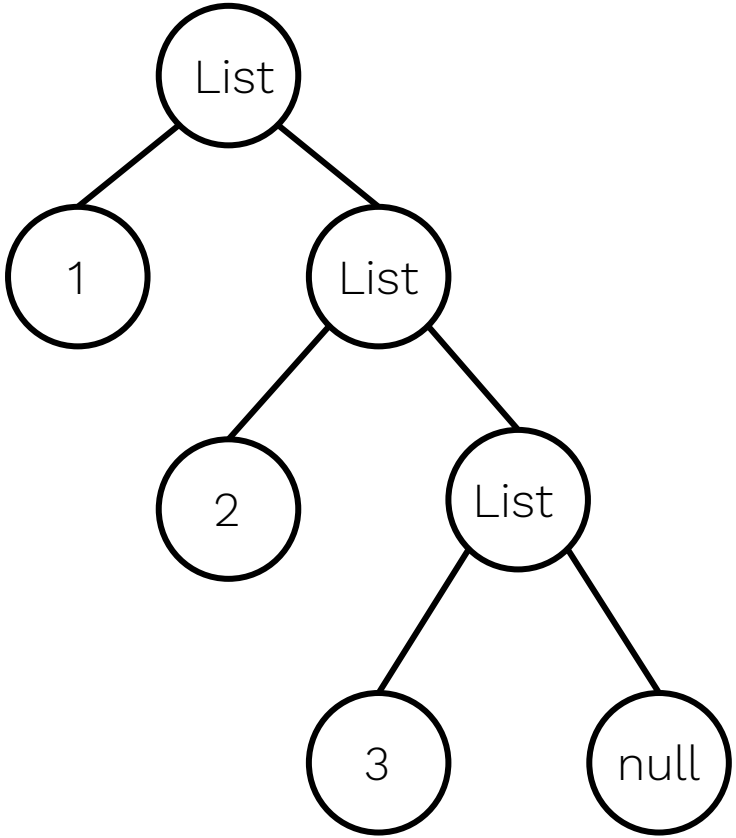

```
function createList(a, b) {
  return function(f) {
    return f(a, b);
  };
}

function first(list) {
  return list(function(a, b) {
    return a;
  });
}

function second(list) {
  return list(function(a, b) {
    return b;
  });
}

const myList = createList('valueA', 'valueB');
const a = first(myList);
const b = second(myList);
console.log(a, b); // valueA, valueB
```



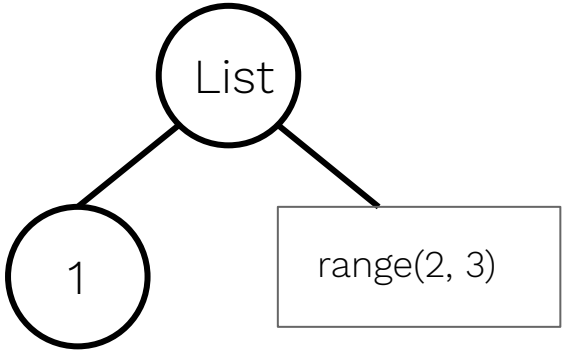

```
function createList(a, b) {
  return function(f) {
    return f(a, b);
  };
}

function first(list) {
  return list(function(a, b) {
    return a;
  });
}

function second(list) {
  return list(function(a, b) {
    return b;
  });
}

const myList = createList('valueA', 'valueB');
const a = first(myList);
const b = second(myList);
console.log(a, b); // valueA, valueB
```

```
19  const myList = createList('A', createList('B', createList('C', 'D')));
20  const a = first(myList);
21  const c = first(second(second(myList)));
22  console.log(a, c); // A, C
```

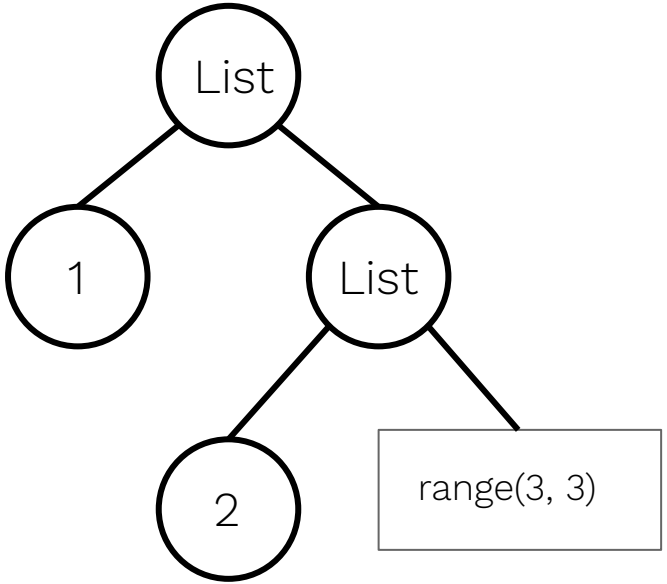




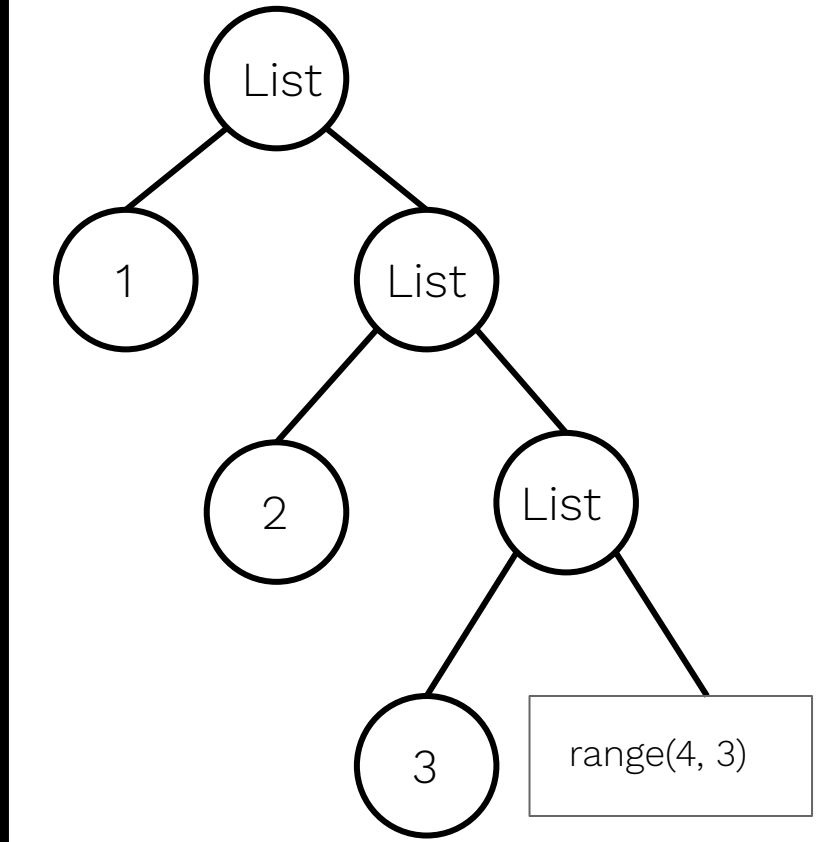

```
function range(from, to) {
  if (from <= to) {
    return createList(from, range(from + 1, to));
  } else {
    return null;
  }
}

const myList = range(1, 3);
const a = first(myList);
const b = first(second(myList));
console.log(a, b); // 1, 2
```



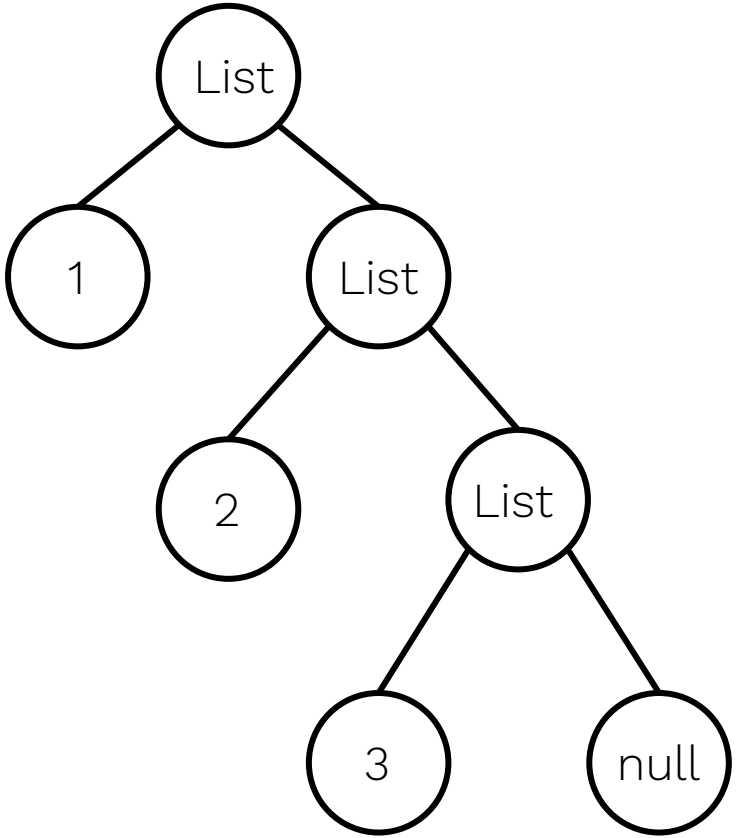

```
function range(from, to) {
  if (from <= to) {
    return createList(from, range(from + 1, to));
  } else {
    return null;
  }
}

const myList = range(1, 3);
const a = first(myList);
const b = first(second(myList));
console.log(a, b); // 1, 2
```



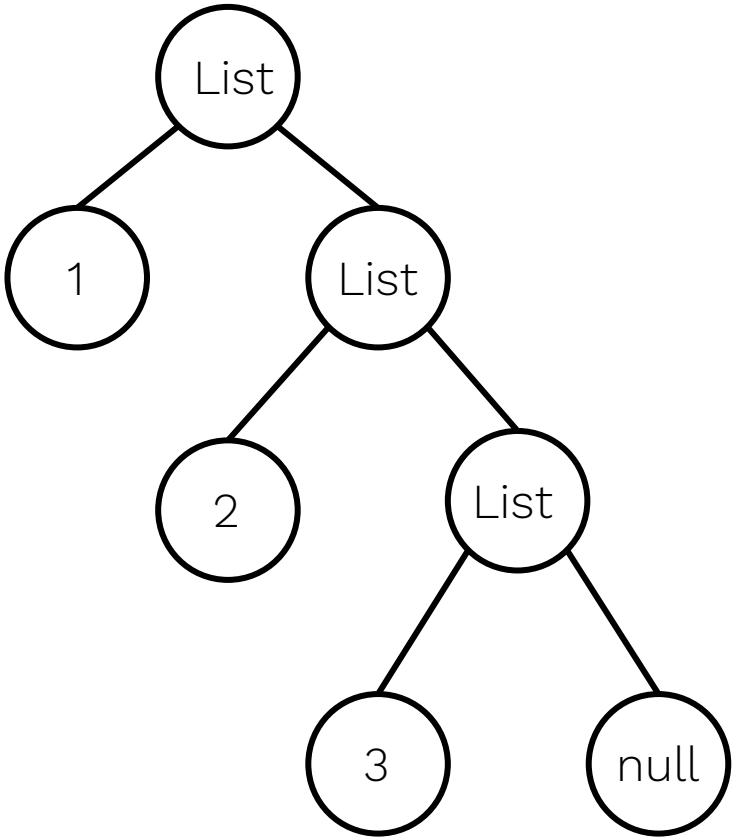

```
function range(from, to) {
  if (from <= to) {
    return createList(from, range(from + 1, to));
  } else {
    return null;
  }
}

const myList = range(1, 3);
const a = first(myList);
const b = first(second(myList));
console.log(a, b); // 1, 2
```



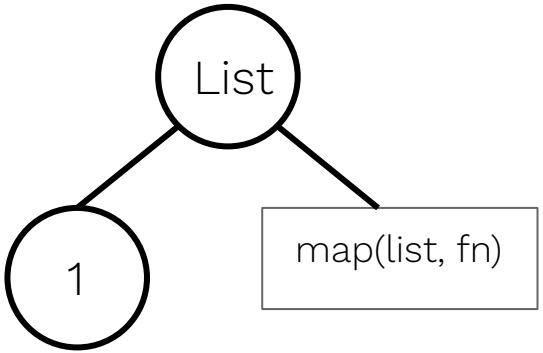

```
function range(from, to) {
  if (from <= to) {
    return createList(from, range(from + 1, to));
  } else {
    return null;
  }
}

const myList = range(1, 3);
const a = first(myList);
const b = first(second(myList));
console.log(a, b); // 1, 2
```



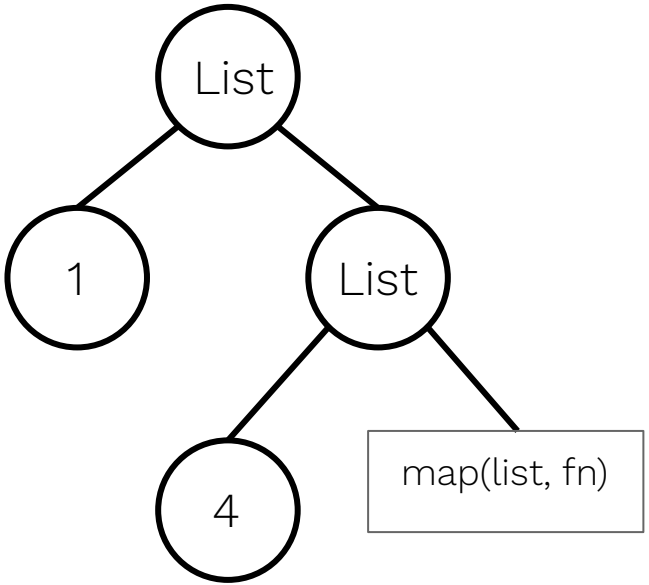

```
function range(from, to) {
  if (from <= to) {
    return createList(from, range(from + 1, to));
  } else {
    return null;
  }
}

const myList = range(1, 3);
const a = first(myList);
const b = first(second(myList));
console.log(a, b); // 1, 2
```



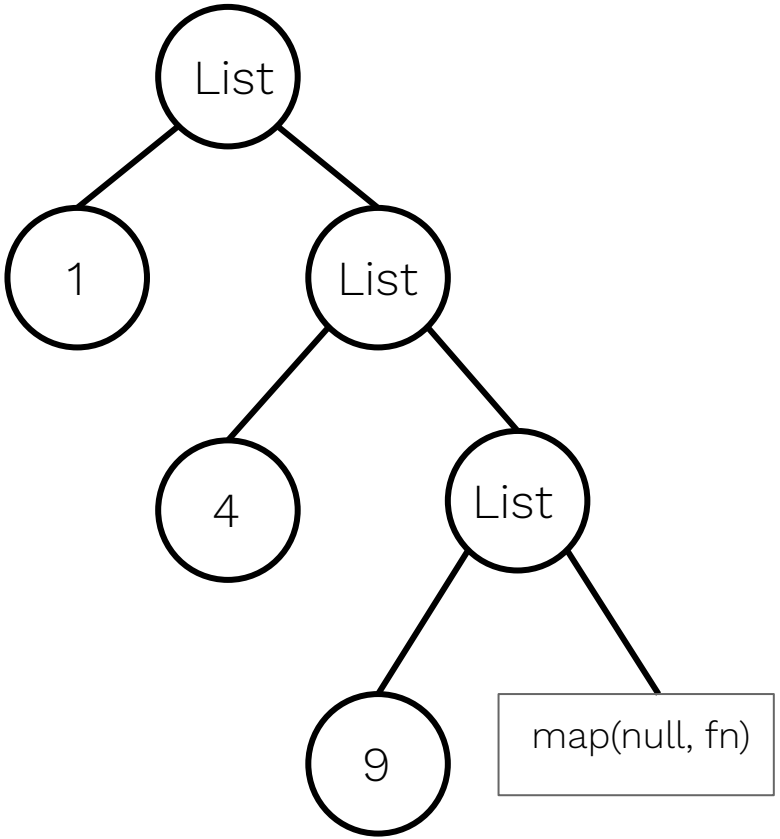

```
function map(list, fn) {
  if (list !== null) {
    return createList(
      fn(first(list)),
      map(second(list), fn)
    );
  } else {
    return list;
  }
}

let myList = range(1, 3);
myList = map(myList, n => n * n);
const a = first(myList);
const b = first(second(myList));
console.log(a, b); // 1, 4
```



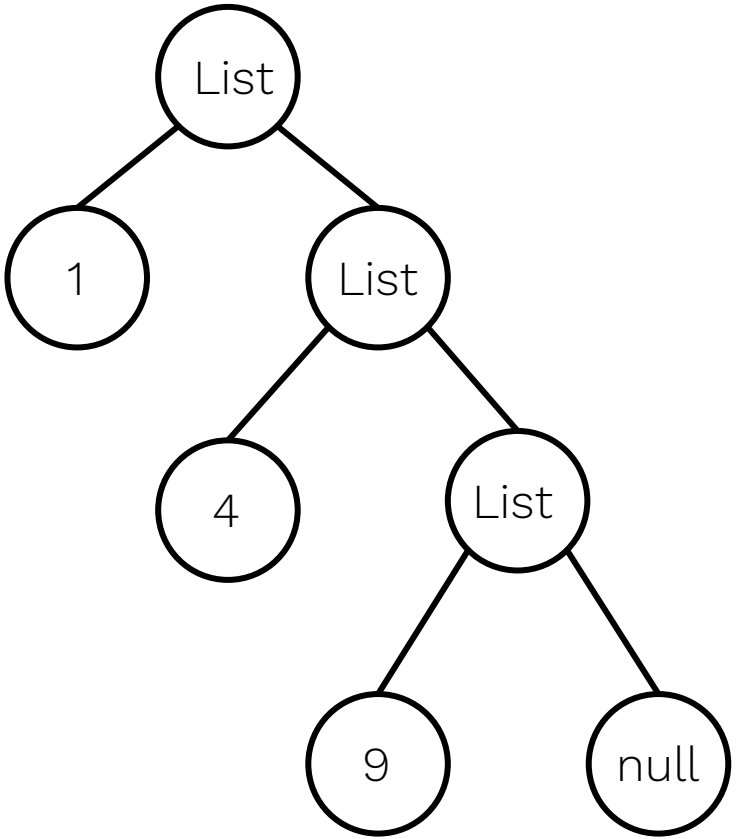

```
function map(list, fn) {
  if (list !== null) {
    return createList(
      fn(first(list)),
      map(second(list), fn)
    );
  } else {
    return list;
  }
}

let myList = range(1, 3);
myList = map(myList, n => n * n);
const a = first(myList);
const b = first(second(myList));
console.log(a, b); // 1, 4
```



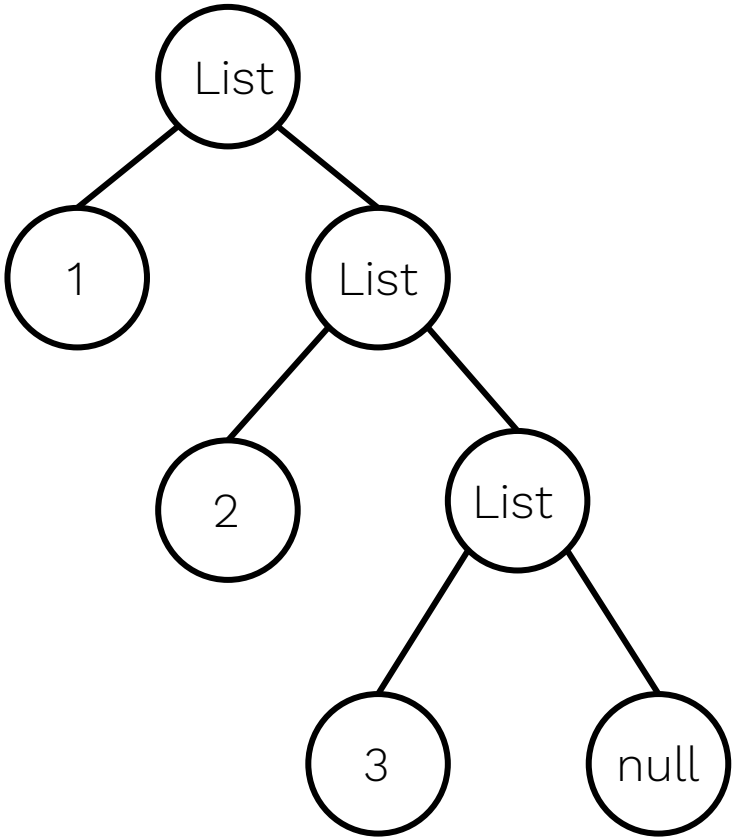

```
function map(list, fn) {
  if (list !== null) {
    return createList(
      fn(first(list)),
      map(second(list), fn)
    );
  } else {
    return list;
  }
}

let myList = range(1, 3);
myList = map(myList, n => n * n);
const a = first(myList);
const b = first(second(myList));
console.log(a, b); // 1, 4
```



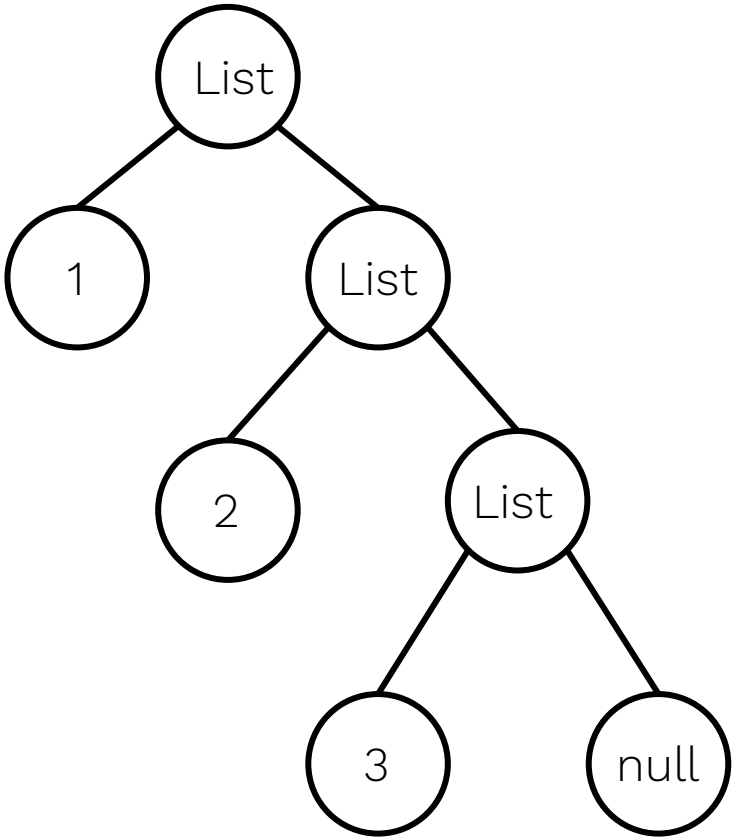

```
function map(list, fn) {
  if (list !== null) {
    return createList(
      fn(first(list)),
      map(second(list), fn)
    );
  } else {
    return list;
  }
}

let myList = range(1, 3);
myList = map(myList, n => n * n);
const a = first(myList);
const b = first(second(myList));
console.log(a, b); // 1, 4
```




```
function map(list, fn) {
  if (list !== null) {
    return createList(
      fn(first(list)),
      map(second(list), fn)
    );
  } else {
    return list;
  }
}

let myList = range(1, 3);
myList = map(myList, n => n * n);
const a = first(myList);
const b = first(second(myList));
console.log(a, b); // 1, 4
```




```
function foreach(list, fn) {
  if (list !== null) {
    fn(first(list));
    foreach(second(list), fn);
  }
}

let numbers = range(1, 3);
foreach(numbers, n => console.log(n));
// 1, 2, 3
```




```
function reverse(list) {
  return function _reverse(list, parent) {
    if (second(list) === null) {
      return createList(first(list), parent);
    } else {
      return _reverse(
        second(list),
        createList(first(list), parent)
      );
    }
  }(list, null);
}
```

```
const list = (a, b) => f => f(a, b);
const first = l => l((a, b) => a);
const second = l => l((a, b) => b);

const range = (from, to) => from <= to ? list(from, range(from + 1, to)) : null;
const foreach = (l, f) => l !== null && (f(first(l)), foreach(second(l), f));
const map = (l, f) => l !== null ? list(f(first(l)), map(second(l), f)) : l;
const reverse = (l, parent = null) =>
  second(l) === null
    ? list(first(l), parent)
    : reverse(second(l), list(first(l), parent));

let numbers = range(1, 10);
numbers = map(numbers, function (n) { return n * n });
numbers = reverse(numbers);
foreach(numbers, (n) => console.log(n));
```

```
function sum(a ,b) {
  return a + b;
}

sum(2, 2);
```

```
function sum(a) {
  return function(b) {
    return a + b;
  }
}

sum(2)(2);
```

```
const list = a => b => f => f(a)(b);
const first = l => l(a => b => a);
const second = l => l(a => b => b);

const range = from => to => from <= to ? list(from)(range(from + 1)(to)) : null;
const foreach = l => f => l !== null && (f(first(l)), foreach(second(l))(f));
const map = l => f => l !== null ? list(f(first(l)))(map(second(l))(f)) : l;
const reverse = l => (parent = null) =>
  second(l) === null
    ? list(first(l))(parent)
    : reverse(second(l))(list(first(l))(parent));

let numbers = range(1)(10);
numbers = map(numbers)(n => n * n);
numbers = reverse(numbers)();
foreach(numbers)(console.log);
```

```
1   const TRUE = t => f => t;
2   const FALSE = t => f => f;
3   const IF_ELSE = c => t => f => c(t)(f)();
4
5   const SMILE = () => console.log(':)');
6   const CRY = () => console.log(';(');
7
8   IF_ELSE(TRUE)
9   (SMILE)
10  (CRY); // :)
11
12  IF_ELSE(FALSE)
13  (SMILE)
14  (CRY); // ;(
```

```
const IDENTITY       = x => x;

const TRUE    = t => f => t;
const AND     = p => q => p(q)(p);
const NOT     = c => c(FALSE)(TRUE);
const IF_ELSE = c => t => f => c(t)(f)();

const SUCCESSOR      = n => f => x => f(n(f)(x));
const PREDECESSOR    = n => f => x => n(g => h => h(g(f)))(_ => x)(u => u);
const ADDITION       = m => n => n(SUCCESSOR)(m);
const SUBTRACTION    = m => n => n(PREDECESSOR)(m);

const IS_ZERO               = n => n(_ => FALSE)(TRUE);
const IS_LESS_THAN_EQUAL    = m => n => IS_ZERO(SUBTRACTION(m)(n));
const IS_EQUAL              = m => n => AND(IS_LESS_THAN_EQUAL(m)(n))(IS_LESS_THAN_EQUAL(n)(m));

const $zero  = f => IDENTITY;
const $one   = SUCCESSOR($zero);
const $two   = SUCCESSOR($one);

const SUCCESS = () => console.log('GREAT SUCCESS!!!');
const FAIL = () => console.log('HUGE FAIL...');

// GREAT SUCCESSS !!!
IF_ELSE(IS_EQUAL($two)(ADDITION($one)($one)))
  (SUCCESS)
  (FAIL);
```

LAMBDA CALCULUS

Rachunek lambda *– system formalny używany do badania zagadnień związanych z podstawami matematyki jak rekurencja, definiowalność funkcji, obliczalność, podstawy matematyki np. definicja liczb naturalnych, wartości logicznych itd. Rachunek lambda został wprowadzony przez Alonzo Churcha i Stephena Cole'a Kleene'ego w 1930 roku.*

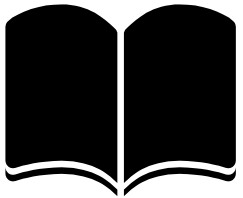
λ-CALCULUS	JAVASCRIPT
`TRUE := λx.λy.x`	`const TRUE = t => f => t;`
`FALSE := λx.λy.y`	`const FALSE = t => f => f;`
`AND := λp.λq.p q p`	`const AND = p => q => p(q)(p);`
`OR := λp.λq.p p q`	`const OR = p => q => p(p)(q);`
`NOT := λp.p FALSE TRUE`	`const NOT = c => c(FALSE)(TRUE);`
`IF_ELSE := λp.λa.λb.p a b`	`const IF_ELSE = c => t => f => c(t)(f)();`

```
1  const Z = f => (x => f(y => (x(x))(y)))(x => f(y => (x(x))(y)));
2
3  const COUNTDOWN = Z(f => i => i >= 0 && (console.log(i), f(i - 1)));
4  COUNTDOWN(10); // 10, 9, 8, 7...
```

LAMBDA CALCULUS
is
turing-complete

```
x => y => x
```

Przydatne linki



- Edytor z przykładami
- Lambda calculus in JS @gtramontina
- Types and Programming Languages
- λ-calculus Interpreter napisany w JS
- Zadanie JS

Q&A

Dzięki!

Kontakt:

- github@krzysztof-grzybek
- twitter@k__grzybek