More functions and recursion

Jia Xiaodong

August 30, 2021

Anonymous functions

```
const g = param => { /* body */ }
```

Anonymous functions

```
const g = param => { /* body */ }
```

Short aside: difference between *parameter* and *argument*:

- A parameter is what the function depends on. For the above, the parameter of `g` is `param`.

Anonymous functions

```
const g = param => { /* body */ }
```

Short aside: difference between *parameter* and *argument*:

- A parameter is what the function depends on. For the above, the parameter of `g` is `param`.
- An argument is what you give the function. For example, `g(5)`, then 5 is the argument.

Higher order functions

Remember this?

```
function fact_helper(n, res) {
    return n === 1
        ? res
        : fact_helper(n - 1, n * res);
}

function factorial(n) {
    return fact_helper(n, 1);
}
```

Higher order functions

Cont.

Makes more sense?

```
function factorial(n) {
    function fact_helper(n, res) {
        return n === 1
            ? res
            : fact_helper(n - 1, n * res);
    }

    return fact_helper(n, 1);
}
```

Higher order functions

Functions as return value

Functions of functions (functionals):

$$I = \int f(t)\,\mathrm{d}t$$

Higher order functions

Functions as arguments

Say I want the smallest of two things.

Higher order functions

Functions as arguments

Say I want the smallest of two things.

```
const min = (a, b) => a < b ? a : b
```

Higher order functions

Functions as arguments

Say I want the smallest of two things.

```
const min = (a, b) => a < b ? a : b
```

What if I am comparing timings in HH:MM format and I want the earliest?

Higher order functions

Functions as arguments

Say I want the smallest of two things.

```
const min = (a, b) => a < b ? a : b
```

What if I am comparing timings in HH:MM format and I want the earliest?

```
const min = (a, b, f) => f(a) < f(b) ? a : b
```

```
function hhmm_to_mins(a) { ... }
min(a, b, hhmm_to_mins);
```

Scoping

- We give names to things.

Scoping

- We give names to things.
- We may give many things the same name. (e.g. c: Speed of light, specific heat capacity, etc.)

Scoping

- We give names to things.
- We may give many things the same name. (e.g. c: Speed of light, specific heat capacity, etc.)
- What gives us the context for our names?

Scopes

- *A name occurrence refers to the closest surrounding declaration.*

Scopes

- *A name occurrence refers to the closest surrounding declaration.*
- Scopes are our context where we find our names.

Scopes

- *A name occurrence refers to the closest surrounding declaration.*
- Scopes are our context where we find our names.
- The most common context are blocks: `{...}`.

Scopes

- *A name occurrence refers to the closest surrounding declaration.*
- Scopes are our context where we find our names.
- The most common context are blocks: `{...}`.
- To find what a name refers to, look at the current scope, and then outwards. Take the first one you come across.

Scopes

- *A name occurrence refers to the closest surrounding declaration.*
- Scopes are our context where we find our names.
- The most common context are blocks: `{...}`.
- To find what a name refers to, look at the current scope, and then outwards. Take the first one you come across.
- Names in an outer scope can be hidden by definitions in an inner scope.

Exercise

```
hello = "world"
function n(hello){
   const g = hello => display;
   g(hello);
}
n("hello")(hello);
```

Exercise

```
hello = "world"
function n(hello){
  const g = hello => display;
  g(hello);
}
n("hello")(hello);
```

```
const n = 1;
{
    const n = 2;
    {
        const n = 3;
        {
            display(n);
        }
        const n = 4;
    }
}
```

Example: series *

Let us make a polynomial series generator. A series is something like

$$S(x) = \sum_{n=0}^{k} a_n x^n = a_0 + a_1 x + a_2 x^2 + \cdots$$

Example: series *

Let us make a polynomial series generator. A series is something like

$$S(x) = \sum_{n=0}^{k} a_n x^n = a_0 + a_1 x + a_2 x^2 + \cdots$$

A disposable solution:

```
function sum(x) {
    return a0 + a1 * x + a2 * x * x + ...
}
```

Example: series *

Cont.

```
function series_generator(k, coeff) {
```

Example: series *

Cont.

```
function series_generator(k, coeff) {
    function gen_helper(n, series) {
        return n === k
            ? series
```

Example: series *

Cont.

```
function series_generator(k, coeff) {
    function gen_helper(n, series) {
        return n === k
            ? series
            : gen_helper(n + 1,
```

Example: series *

Cont.

```
function series_generator(k, coeff) {
    function gen_helper(n, series) {
        return n === k
            ? series
            : gen_helper(n + 1,
                x => series(x) + coeff(n) * math_pow(x, n))
```

Example: series *

Cont.

```
function series_generator(k, coeff) {
    function gen_helper(n, series) {
        return n === k
            ? series
            : gen_helper(n + 1,
                x => series(x) + coeff(n) * math_pow(x, n))
    }
```

Example: series *

Cont.

```
function series_generator(k, coeff) {
    function gen_helper(n, series) {
        return n === k
            ? series
            : gen_helper(n + 1,
                x => series(x) + coeff(n) * math_pow(x, n))
    }
    return gen_helper(0, x => 0);
}
```

Example: series *

Demonstration

$$e^x \approx \sum_{n=0}^{k} \frac{x^n}{n!}$$

Example: series *

Demonstration

$$e^x \approx \sum_{n=0}^{k} \frac{x^n}{n!}$$

```
function exp_coeff(n) {
    return 1 / factorial(n);
}
const exp_series = series_generator(5, exp_coeff);
```

Try it out!

Example: series *

Demonstration, cont.

$$\sin(x) \approx \sum_{n=0}^{k} \frac{(-1)^n x^{2n+1}}{(2n+1)!}$$

Example: series *

Demonstration, cont.

$$\sin(x) \approx \sum_{n=0}^{k} \frac{(-1)^n x^{2n+1}}{(2n+1)!}$$

```
function sin_coeff(n) {
    function minus_one(n) {
        return ((n - 1) / 2) % 2 === 0 ? 1 : -1;
    }
    return n % 2 === 0 ? 0 : minus_one(n) / factorial(n);
}
const sin_series = series_generator(5, sin_coeff);
```

Try it out! (same link as before)

Example: series *

Challenge

Fourier trigonometric series for function f with period $2L$:

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \sin\left(\frac{n\pi x}{L}\right) + \sum_{n=1}^{\infty} b_n \cos\left(\frac{n\pi x}{L}\right)$$

Primitive recursion *

Definition

The following are primitive recursive:

- Constant function: 0
- Successor function: $S(x) = x + 1$
- Projection function[1]: $P_i(\mathbf{x}) = \mathbf{x}_i$

Recursion: if f, g are primitive recursive, h is primitive recursive if

$$h(0, \mathbf{x}) = f(\mathbf{x})$$
$$h(S(y), \mathbf{x}) = g(y, h(y, \mathbf{x}), \mathbf{x})$$

[1] In subsequent slides $\mathbf{x}$ is the vector of arguments given to the function (i.e. represents $x_1, x_2, \ldots$), and $\mathbf{x}_i$ is the i-th element of the vector (i.e. x_i).

Primitive recursion *

Definition

A function f is defined from t by *iteration* if

$$f(\mathbf{x}, n) = t^n(\mathbf{x})$$

Theorem

Minus some formalities, primitive recursion and iteration are equivalent.

Proof.

Iteration is primitive recursion because

$$f(\mathbf{x}, 0) = \mathbf{x}$$
$$f(\mathbf{x}, n+1) = t(f(\mathbf{x}, n)) \qquad \square$$

Primitive recursion *

Proof.

Primitive recursion can be converted into recursion. Take

$$t(\mathbf{x}, n, z) := (\mathbf{x}, n+1, h(\mathbf{x}, n, z))$$

Then

$$(\mathbf{x}, n, f(\mathbf{x}, n)) = t^n(\mathbf{x}, 0, g(\mathbf{x}))$$

□

Example: factorial

Factorial is defined as follows:

$$f(0) = g := 1$$
$$f(n+1) = h(n, f(n)) := (n+1) \cdot f(n)$$

Primitive recursion *

Example: factorial

Let us make it iterative. Then using the recipe,

$$t(n, z) := (n + 1, (n + 1) \cdot z)$$
$$(n + 1, n!) = t^n(0, 1)$$

S4 Q1

Write a function computing elements of Pascal's triangle, i.e. $\binom{\text{row}}{\text{col}}$.
The following relationships might be helpful:

$$\binom{r}{c} = \binom{r-1}{c-1} + \binom{r-1}{c} \qquad \binom{r}{1} = \binom{r}{r} = 1$$

S4 Q1

Write a function computing elements of Pascal's triangle, i.e. $\binom{\text{row}}{\text{col}}$.
The following relationships might be helpful:

$$\binom{r}{c} = \binom{r-1}{c-1} + \binom{r-1}{c} \qquad \binom{r}{1} = \binom{r}{r} = 1$$

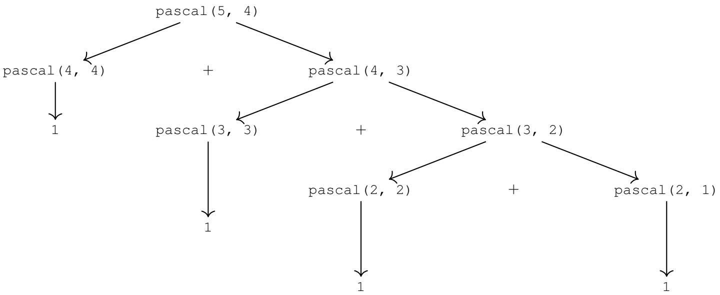
```
function pascal(row, col) {
    return col === 1 || col === row
        ? 1
        : pascal(row - 1, col - 1) + pascal(row - 1, col);
}
```

S4 Q2

Draw the tree illustration the process generated by `pascal(5, 4)`.

S4 Q2

Draw the tree illustration the process generated by `pascal(5, 4)`.



Recursive.

S4 IC-Q1

What do the following evaluate to?

```
compose(math_sqrt, math_log)(math_E)
compose(math_log, math_sqrt)(math_E * math_E)
```

S4 IC-Q1

What do the following evaluate to?

```
compose(math_sqrt, math_log)(math_E)
compose(math_log, math_sqrt)(math_E * math_E)
```

```
(z => math_sqrt(math_log(z)))(math_E)
```

S4 IC-Q1

What do the following evaluate to?

```
compose(math_sqrt, math_log)(math_E)
compose(math_log, math_sqrt)(math_E * math_E)
```

```
(z => math_sqrt(math_log(z)))(math_E)
```

```
(y => math_log(math_sqrt(z)))(math_E * math_E)
```

S4 IC-Q2

```
const compose = (f, g) => x => f(g(x));
function thrice(f) {
    return compose(compose(f, f), f);
}
```

S4 IC-Q2

```
const compose = (f, g) => x => f(g(x));
function thrice(f) {
    return compose(compose(f, f), f);
}
```

```
thrice(h);
compose(compose(h, h), h)
```

S4 IC-Q2

```
const compose = (f, g) => x => f(g(x));
function thrice(f) {
    return compose(compose(f, f), f);
}
```

```
thrice(h);
compose(compose(h, h), h)
compose(x => h(h(x)), h)
```

S4 IC-Q2

```
const compose = (f, g) => x => f(g(x));
function thrice(f) {
    return compose(compose(f, f), f);
}
```

```
thrice(h);
compose(compose(h, h), h)
compose(x => h(h(x)), h)
y => (x => h(h(x))(h(y))
```

S4 IC-Q2

```
const compose = (f, g) => x => f(g(x));
function thrice(f) {
    return compose(compose(f, f), f);
}
```

```
thrice(h);
compose(compose(h, h), h)
compose(x => h(h(x)), h)
y => (x => h(h(x))(h(y))

thrice(h)(z);
(x => h(h(x))(h(z))
h(h(h(z)))
```

S4 IC-Q3

```
const compose = (f, g) => x => f(g(x));
function repeated(f, n) {
    return n === 0
        ? x => x
        : compose(f, repeated(f, n - 1));
}
```

S4 IC-Q3

```
const compose = (f, g) => x => f(g(x));
function repeated(f, n) {
    return n === 0
        ? x => x
        : compose(f, repeated(f, n - 1));
}
```

```
repeated(f, 2);
compose(f, repeated(f, 1))
```

S4 IC-Q3

```
const compose = (f, g) => x => f(g(x));
function repeated(f, n) {
    return n === 0
        ? x => x
        : compose(f, repeated(f, n - 1));
}
```

```
repeated(f, 2);
compose(f, repeated(f, 1))
compose(f, compose(f, repeated(f, 0)))
```

S4 IC-Q3

```
const compose = (f, g) => x => f(g(x));
function repeated(f, n) {
    return n === 0
        ? x => x
        : compose(f, repeated(f, n - 1));
}
```

```
repeated(f, 2);
compose(f, repeated(f, 1))
compose(f, compose(f, repeated(f, 0)))
compose(f, compose(f, x => x))
```

S4 IC-Q3

```
const compose = (f, g) => x => f(g(x));
function repeated(f, n) {
    return n === 0
        ? x => x
        : compose(f, repeated(f, n - 1));
}
```

```
repeated(f, 2);
compose(f, repeated(f, 1))
compose(f, compose(f, repeated(f, 0)))
compose(f, compose(f, x => x))
compose(f, y => f((x => x)(y)))
```

S4 IC-Q3

```
const compose = (f, g) => x => f(g(x));
function repeated(f, n) {
    return n === 0
        ? x => x
        : compose(f, repeated(f, n - 1));
}
```

```
repeated(f, 2);
compose(f, repeated(f, 1))
compose(f, compose(f, repeated(f, 0)))
compose(f, compose(f, x => x))
compose(f, y => f((x => x)(y)))
z => f((y => f((x => x)(y)))(z))
```

S4 IC-Q3

```
const compose = (f, g) => x => f(g(x));
function repeated(f, n) {
    return n === 0
        ? x => x
        : compose(f, repeated(f, n - 1));
}
```

```
repeated(f, 2);
compose(f, repeated(f, 1))
compose(f, compose(f, repeated(f, 0)))
compose(f, compose(f, x => x))
compose(f, y => f((x => x)(y)))
z => f((y => f((x => x)(y)))(z))

repeated(f, 2)(a);
f((y => f((x => x)(y)))(a))
```

S4 IC-Q3

```
const compose = (f, g) => x => f(g(x));
function repeated(f, n) {
    return n === 0
        ? x => x
        : compose(f, repeated(f, n - 1));
}
```

```
repeated(f, 2);
compose(f, repeated(f, 1))
compose(f, compose(f, repeated(f, 0)))
compose(f, compose(f, x => x))
compose(f, y => f((x => x)(y)))
z => f((y => f((x => x)(y)))(z))

repeated(f, 2)(a);
f((y => f((x => x)(y)))(a))
f((f((x => x)(a))))
```

S4 IC-Q3

```
const compose = (f, g) => x => f(g(x));
function repeated(f, n) {
    return n === 0
        ? x => x
        : compose(f, repeated(f, n - 1));
}
```

```
repeated(f, 2);
compose(f, repeated(f, 1))
compose(f, compose(f, repeated(f, 0)))
compose(f, compose(f, x => x))
compose(f, y => f((x => x)(y)))
z => f((y => f((x => x)(y)))(z))

repeated(f, 2)(a);
f((y => f((x => x)(y)))(a))
f((f((x => x)(a))))
f((f((a))))
```

S4 IC-Q3

Cont.

```
const compose = (f, g) => x => f(g(x));
function thrice(f) {
    return compose(compose(f, f), f);
}
```

For what value of `n` will `((thrice(thrice))(f))(0)` return the same value as `(repeated(f, n))(0)`?

S4 IC-Q3

Cont.

```
const compose = (f, g) => x => f(g(x));
function thrice(f) {
    return compose(compose(f, f), f);
}
```

For what value of `n` will `((thrice(thrice))(f))(0)` return the same value as `(repeated(f, n))(0)`?

```
// thrice(h)(z) ---> h(h(h(z)))
(thrice(thrice))(f);
```

S4 IC-Q3

Cont.

```
const compose = (f, g) => x => f(g(x));
function thrice(f) {
    return compose(compose(f, f), f);
}
```

For what value of `n` will `((thrice(thrice))(f))(0)` return the same value as `(repeated(f, n))(0)`?

```
// thrice(h)(z) ---> h(h(h(z)))
(thrice(thrice))(f);
thrice(thrice(thrice(f)))
```

S4 IC-Q3

Cont.

```
const compose = (f, g) => x => f(g(x));
function thrice(f) {
    return compose(compose(f, f), f);
}
```

For what value of `n` will `((thrice(thrice))(f))(0)` return the same value as `(repeated(f, n))(0)`?

```
// thrice(h)(z) ---> h(h(h(z)))
(thrice(thrice))(f);
thrice(thrice(thrice(f)))

((thrice(thrice))(f))(0);
(thrice(thrice(thrice(f))))(0)
```

S4 IC-Q3

Cont.

```
const compose = (f, g) => x => f(g(x));
function thrice(f) {
    return compose(compose(f, f), f);
}
```

For what value of `n` will `((thrice(thrice))(f))(0)` return the same value as `(repeated(f, n))(0)`?

```
// thrice(h)(z) ---> h(h(h(z)))
(thrice(thrice))(f);
thrice(thrice(thrice(f)))

((thrice(thrice))(f))(0);
(thrice(thrice(thrice(f))))(0)
g(g(g(0))) // g = thrice(thrice(f))
```

S4 IC-Q3

Cont.

```
const compose = (f, g) => x => f(g(x));
function thrice(f) {
    return compose(compose(f, f), f);
}
```

For what value of `n` will `((thrice(thrice))(f))(0)` return the same value as `(repeated(f, n))(0)`?

```
// thrice(h)(z) ---> h(h(h(z)))
(thrice(thrice))(f);
thrice(thrice(thrice(f)))

((thrice(thrice))(f))(0);
(thrice(thrice(thrice(f))))(0)
g(g(g(0))) // g = thrice(thrice(f))
g(g(h(h(h(0))))) // h = thrice(f)
```

S4 IC-Q3

Cont.

```
const compose = (f, g) => x => f(g(x));
function thrice(f) {
    return compose(compose(f, f), f);
}
```

For what value of `n` will `((thrice(thrice))(f))(0)` return the same value as `(repeated(f, n))(0)`?

```
// thrice(h)(z) ---> h(h(h(z)))
(thrice(thrice))(f);
thrice(thrice(thrice(f)))

((thrice(thrice))(f))(0);
(thrice(thrice(thrice(f))))(0)
g(g(g(0))) // g = thrice(thrice(f))
g(g(h(h(h(0))))) // h = thrice(f)
g(g(h(h(f(f(f(0))))))))
```

S4 IC-Q3

Cont.

```
// thrice(h)(z) ---> h(h(h(z)))
((thrice(thrice))(f))(0);
(thrice(thrice(thrice(f))))(0)
g(g(g(0)))              // g = thrice(thrice(f))
g(g(h(h(h(0)))))        // h = thrice(f)
g(g(h(h(f(f(f(0)))))))
g(g(h(f(f(f(a)))))) // a = f(f(f(0)))
g(g(f(f(f(b)))))        // b = f(f(f(a)))
g(g(c))                 // c = f(f(f(b))) = ffffffa = fffffffff0
g(fffffffffc)
fffffffffd              // d = fffffffffc = fffffffff fffffffff0
fffffffff fffffffff fffffffff0
```

27.

S4 IC-Q4a

```
((thrice(thrice))(add1))(6);
```

S4 IC-Q4a

```
((thrice(thrice))(add1))(6);
```

```
aaaaaaaaa aaaaaaaaa aaaaaaaaa6
```

33.

S4 IC-Q4b

```
((thrice(thrice))(x => x))(compose);
```

S4 IC-Q4b

```
((thrice(thrice))(x => x))(compose);
```

```
fffffffff fffffffff fffffffffc
```

S4 IC-Q4b

```
((thrice(thrice))(x => x))(compose);
```

```
fffffffff fffffffff fffffffffc
fffffffff fffffffff ffffffffc
```

S4 IC-Q4b

```
((thrice(thrice))(x => x))(compose);
```

```
fffffffff fffffffff fffffffffc
fffffffff fffffffff ffffffffc
c
```

S4 IC-Q4c,d

```
((thrice(thrice))(square))(2);
```

S4 IC-Q4c,d

```
((thrice(thrice))(square))(2);
```

```
sssssssss sssssssss sssssssss2  // 2^1
```

S4 IC-Q4c,d

```
((thrice(thrice))(square))(2);
```

```
sssssssss sssssssss sssssssss2  // 2^1
sssssssss sssssssss ssssssss4   // 2^2
```

S4 IC-Q4c,d

```
((thrice(thrice))(square))(2);
```

```
sssssssss sssssssss sssssssss2  // 2^1
sssssssss sssssssss ssssssss4   // 2^2
sssssssss sssssssss sssssss16   // 2^4
```

S4 IC-Q4c,d

```
((thrice(thrice))(square))(2);
```

```
sssssssss sssssssss sssssssss2   // 2^1
sssssssss sssssssss ssssssss4    // 2^2
sssssssss sssssssss sssssss16    // 2^4
```

$2^{2^{27}} \gtrapprox 2^{\text{100 million}}$