Revision - Recursion

```
//An example of iteration
function factorial(n){
    return iter(1,1,n);
}
function iter(product,conter,n){
    return counter>n ? product : iter(counter*product , counter+1 ,n);
}
```

Exercise 1

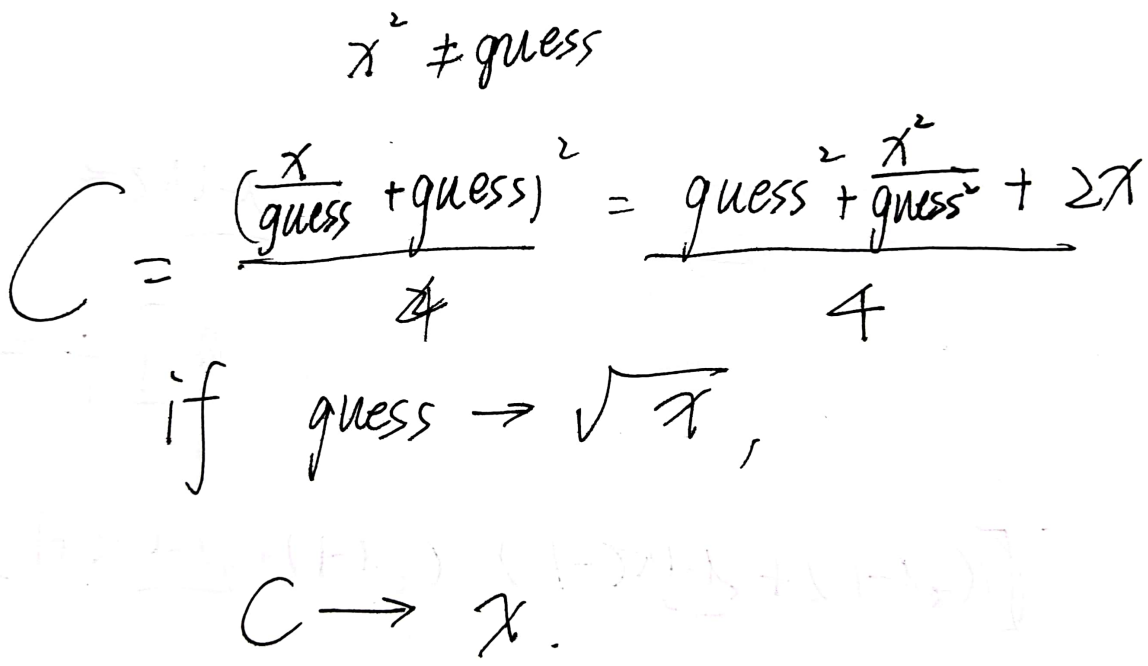
```
//Graph Recursion:
https://github.com/IssacLO4/NUS-SICP/tree/master/docs_out
```

```
//pow in decision of parity
function exp(m,n){
    if(n===0){
        return 1;
    }
    else{
        return m*exp(m,n-1);
    }
}
exp(2,12);
function exp1(m,n){
    if(n===0){
        return 1;
    }
    else{
        if(n%2===0){
            return square(exp1(m,n/2));
        }
        else{
            return m*exp1(m,n-1);
        }
    }

}
exp1(2,12);
```

Exercise 2

```
//SquareRoot Iteration
function abs(x) {
    return x >= 0 ? x : - x;
}

function square(x) {
    return x * x;
```

```
}

function is_good_enough(guess, x) {
    return abs(square(guess)-x)<0.000001;
}

function average(x, y) {
    return (x + y) / 2;
}

function improve(guess, x) {
    return average(guess,x/guess);
}

function sqrt_iter(guess, x) {
    return is_good_enough(guess, x)
           ? guess
           : sqrt_iter(improve(guess, x), x);
}

sqrt_iter(3, 25);
```



Higher-order Functions

Nested function declarations

Functions

```
function hypotunese(a,b){
    function sum_square(){
        return square(a)+square(b);
    }
    return math_sqrt(sum_square()) ;
}
//This implies how a function can be placed inside another one with relevant
parameters needed
```

Functions as arguments

```
//Use the function name as parameters
function f(g,x){
    return g(x);
}
function h(y){
    return y*y;
}
f(h,7);
```

```
function my_repeat(n,f,x){
    return n===0
           ?x
           :my_repeat(n,f,f(x));//Use functions as parameters
    //     :f(my_repeat(n,f,x));//Use recursions
}
function f(x){
    return x+1;
}
```

Lambda functions

```
//Example 1 ,with recursion
function sum(term,a,next,b){
    return a>b
           ?0
           :term(a)+sum(term,next(a),next,b);
}
function sum_cubes(a,b){
    return sum(x=>x*x*x,a,x=>x+2,b);
}
//Example 2,returning function from function
function adder(x){
    return y => x+y;
}
const adder_4 = adder(4);
//adder(4)(6);
adder_4(6);//==10
```

Exercises

```
function sum(term, a, next, b) {
    return a > b
       ? 0
       : term(a) + sum(term, next(a), next, b);
}
//Using sum, create a function sum_odd that produces the sum of the first n odd
natural numbers, n >= 0. You can assume that the function sum is already
declared. You do not need to implement it yourself.

function term(n) {
  return 2 * n - 1;  // Calculates the nth odd number
}

function next(n) {
  return n + 1;  // Increment to the next term
}

function sum_odd(n) {
  return sum(term, 1, next, n);
}
//注意到在javascript中，定义n为const变量之后不可以使用表达式（比如将2n-1放入函数）
```

```
//Using sum, create a function sum_odd_lte(n) that produces the sum of all odd
natural numbers less than or equal to n. You can assume that the function sum is
already declared. You do not need to implement it yourself.
function term(n) {
  return n;  // Calculates the nth odd number
}

function next(n) {
  return n + 2;  // Increment to the next term
}

function sum_odd_lte(n) {
  return sum(term, 1, next, n);
}
```

https://sourceacademy.nus.edu.sg/courses/62/paths/839/2

```
//consider a very similar function product.
function product(term, a, next, b) {
    return a > b
       ? 1
       : term(a) * product(term, next(a), next, b);
}

// Example usage:

function fact(n) {
    return product(x => x, 1, x => x + 1, n);
}
```

```
//We can generalize this into a function accumulate(combiner, term, a, next, b,
base) such that we can express sum and product as

function sum(term, a, next, b) {
   return accumulate( (x, y) => x + y, term, a, next, b, 0);
}

function product(term, a, next, b) {
   return accumulate( (x, y) => x * y, term, a, next, b, 1);
}
//Implement the function accumulate(combiner, term, a, next, b, base) that does
this.
//Hint: modify sum.
//Answer:
function accumulate(combiner, term, a, next, b, base) {
  return a > b
    ? base
    : combiner(term(a), accumulate(combiner, term, next(a), next, b, base));
}
```

Getting Started with Curves

Functions

```
//Coordinates
x_of();y_of();z_of()
//return the particular coordinates

//connecting curves
connect_ends(curve1: Curve, curve2: Curve): Curve
//将两条曲线头尾相接；The two Curves are combined by using the full first Curve for
the first portion of the result and by using the full second Curve for the second
portion of the result. The second Curve is translated such that its point at
fraction 0 is the same as the Point of the first Curve at fraction 1.
connect_rigidly(curve1: Curve, curve2: Curve): Curve
//将两条曲线连接，但不改变第二条曲线的位置（当两条曲线不相连时中间会有空当）The two Curves
are combined by using the full first Curve for the first portion of the result
and by using the full second Curve for the second portion of the result. The
second Curve is not changed, and therefore there might be a big jump in the
middle of the result Curve.

//Presenting Curves
draw_connected_full_view_proportional(numPoints: number): RenderFunction
//Returns a function that turns a given Curve into a Drawing, by sampling the
Curve at num sample points and connecting each pair with a line. The Drawing is
translated and scaled proportionally to show the full curve and maximize its
size, with some padding.
//Example:
raw_connected_full_view_proportional(100)(t => make_point(t, t));

//Inverting Curves
invert(curve: Curve): Curve
```

```
//将曲线从尾到头翻转；This function is a Curve transformation: a function from a Curve to a Curve. The points of the result Curve are the same points as the points of the original Curve, but in reverse: The result Curve applied to 0 is the original Curve applied to 1 and vice versa.

//Making Curves
make_point(x: number, y: number): Point
//Example:
const point = make_point(0.5, 0.5);
//Makes a Point with given x and y coordinates.当在函数中使用时则自动从(0,0)持续到(x,x)，不需要循环结构即可划线
translate(x0: number, y0: number, z0: number): CurveTransformer
//从指定的坐标位置绘制已存在的曲线；This function returns a Curve transformation: It takes an x-value x0, a y-value y0 and a z-value z0, as arguments and returns a Curve transformation that takes a Curve as argument and returns a new Curve, by translating the original by x0 in x-direction, y0 in y-direction and z0 in z-direction.
```

Exercises

Flipping curves

https://sourceacademy.nus.edu.sg/courses/62/missions/840/0

```
function reflect_through_y_axis(curve) {
    return t => make_point(-x_of(curve(t)),y_of(curve(t)));
    // your answer here
}

// testing using S-curve

function three_quarters(pt) {
    return t => make_point(x_of(pt) + math_cos(1.5 * math_PI * t),
                           y_of(pt) + math_sin(1.5 * math_PI * t));
}

function three_quarters_inverted_and_reversed(pt) {
    return t => make_point(x_of(pt) - math_cos(1.5 * math_PI * (1 - t)),
                           y_of(pt) - math_sin(1.5 * math_PI * (1 - t)));
}

function s_generator(pt) {
    return t =>
         t < 0.5
         ? (three_quarters(make_point(x_of(pt), y_of(pt) + 1)))(2 * t)
         : (three_quarters_inverted_and_reversed(
               make_point(x_of(pt), y_of(pt) - 1)))(2 * (t - 0.5));
}

// the s-curve
const my_s_curve = s_generator(make_point(0.5, 0.25));

// reflecting the s-curve
 draw_connected_full_view_proportional(200)(my_s_curve);
```

```
// reflecting the s-curve
draw_connected_full_view_proportional(200)(reflect_through_y_axis(my_s_curve));
```

Closing curves to connect other ones

https://sourceacademy.nus.edu.sg/courses/62/missions/840/1

```
// you can define your own helper functions here

function close(curve) {
    return connect_ends(curve,invert(curve));
    // your answer here
}

function reflect_through_y_axis(curve) {
    return t => make_point(-x_of(curve(t)),y_of(curve(t)));
    // your answer here
}


// testing using S-curve

function three_quarters(pt) {
    return t => make_point(x_of(pt) + math_cos(1.5 * math_PI * t),
                           y_of(pt) + math_sin(1.5 * math_PI * t));
}

function three_quarters_inverted_and_reversed(pt) {
    return t => make_point(x_of(pt) - math_cos(1.5 * math_PI * (1 - t)),
                           y_of(pt) - math_sin(1.5 * math_PI * (1 - t)));
}

function s_generator(pt) {
    return t =>
         t < 0.5
         ? (three_quarters(make_point(x_of(pt), y_of(pt) + 1)))(2 * t)
         : (three_quarters_inverted_and_reversed(
                make_point(x_of(pt), y_of(pt) - 1)))(2 * (t - 0.5));
}

// the s-curve
const my_s_curve = s_generator(make_point(0.5, 0.25));

 draw_connected_full_view_proportional(200)(my_s_curve);

 draw_connected_full_view_proportional(200)(close(my_s_curve));
// connecting reflection without closing

// connection reflection with closing
draw_connected_full_view_proportional(200)
    (connect_ends(close(my_s_curve), reflect_through_y_axis(my_s_curve)));
```

Drawing Squares/Circles/Lines

https://sourceacademy.nus.edu.sg/courses/62/quests/838/0

```
//Squares
function unit_square(t) {
  if (t <= 1/4) {
    return make_point(4 * t, 0);
  } else if (t <= 1/2) {
    return make_point(1, 4 * (t - 1/4));
  } else if (t <= 3/4) {
    return make_point(1 - 4 * (t - 1/2), 1);
  } else {
    return make_point(0, 1 - 4 * (t - 3/4));
  }
}

// Test
draw_points_full_view_proportional(80)(unit_square);
```

```
//Lines
function diagonal(t) {
    return make_point(t,t);
}

// Test
draw_points(50)(diagonal);
```

```
//Circles
function unit_circle(t) {
    return make_point(math_cos(2 * math_PI * t),
                      math_sin(2 * math_PI * t));
}
```

Curves Advance

Spiral

```
function unit_circle(t) {
    return make_point(math_cos(2 * math_PI * t),
                      math_sin(2 * math_PI * t));
}
//one spiral
function spiral_one(t) {
    const p = unit_circle(t);
    return make_point(t * x_of(p), t * y_of(p));
}
draw_connected_full_view_proportional(200)(spiral_one);

//spiral round(multiple)
function spiral(rev, t) {
    const p = unit_circle((t * rev) % 1);
    return make_point(t * x_of(p), t * y_of(p));
```

```
}
function spiral(rev) {
    return t => {
    const p = unit_circle((t * rev) % 1);
    return make_point(t * x_of(p), t * y_of(p));
    };
}
```

Transformations

```
//Two ways of realizing curves in angles and positions
const rot_line = rotate_around_origin(0,0,math_PI/6)(unit_line);
const shifted_rot_line = translate(0,0,0,25)(rot_line);
draw_connected(200)(shifted_rot_line);

//Using COMPOSE
function compose(f,g){
    return x => f(g(x));
}
const shift_rot = compose(translate(0, 0.25, 0),
                          rotate_around_origin(0, 0, math_PI / 6));
const shifted_rot_line = shift_rot(unit_line);
draw_connected(200)(shifted_rot_line);
```

Connections

```
function connect_rigidly(curve1,curve2){
    return t =>t < 1/2
           ? curve1(2*t)
           : curve(2*t-1);
}
const result_curve = connect_rigidly(arc,translate(1,0,0)(arc));
draw_connected_full_view_proportional(200)(result_curve);
```

Colors

```
function colorful_spiral(rev) {
    return t => {
        const p = unit_circle((t * rev) % 1);
        const R = math_max(0, 1 - 2 * t) * 255;
        const G = (1 - math_abs(1 - 2 * t)) * 255;
        const B = math_max(0, 2 * t - 1) * 255;
        return make_color_point(t * x_of(p), t * y_of(p),R, G, B);
    };
}
draw_connected_full_view_proportional(2000)(colorful_spiral(33));
```

Data

Pair

```
//Definition 1
function pair(x,y){
    return component => component === 0 ? x : y;
}
function head(p){
    return p(0);
}
function tail(p){
    return p(1);
}
//Definition 2
const pair = (x,y) => f => f(x,y);
const head = p => p((x,y)=>x);
const tail = p => p((x,y)=>y);
```

Rational Number

```
function make_rat(n,d){
    return pair(n,d);//初始化分数
}
function numer(x){
    return head(x);//返回分子x
}
function denom(x){
    return tail(x);//返回分母y
}
function  add_rat(x,y){
    return make_rat((numer(x) * denom(y) + numer(y) * denom(x),denom(x) *
denom(y));)//分数相加
}
function  sub_rat(x,y){
    return make_rat((numer(x) * denom(y) + numer(y) * denom(x),denom(x) *
denom(y));//分数相减
}
function  mul_rat(x,y){
    return make_rat(numer(x)*numer(y),denom(x)*denom(y));//分数相乘
}
function div_rat(x,y){
    return make_rat(numer(x)*numer(y),denom(x)*numer(y));//分数相除
}
```

```
//判断分数是否相等
function equal_rat(x,y){
    return numer(X)*denom(y) === numer(y)*denom(x);
}
```

```
//变换分数形式为字符串
function rat_to_string(x){
    return stringify(numer(x))+"/"stringify(denom(x));
}
```

```
//最大公约数
function gcd(a,b){
    return b === 0 ? a :gcd(a%b);
}
function make_rat(n,d){
    const g= gcd(n,d);
    return pair(n/g,d/g);
}
```

List

Make sure that head(p) always has the **data**, and tail(p) always has the **remaining elements**.

Use **NULL** to represent empty lists.

- `pair(x, y)` — returns pair made of `x` and `y`
- `is_pair(p)` — returns `true` iff `p` is a pair
- `null` — represents an empty list
- `is_null(xs)` — returns `true` iff `xs` is the empty list `null`
- `head(p)` — returns the head (first component) of the pair `p`
- `tail(p)` — returns the tail (second component) of the pair `p`
- `list(x1,...,xn)` — returns a list whose first element is `x1`, second element is `x2`, etc. and last element is `xn`
- ...

pair(x,y) => [x,y]

- In ***box notation***
  - `pair(x, y)` is printed as `[x, y]`
  - Empty lists are printed as `null`

- **Example:**

```
pair(1, pair(2, pair(3, null)));
```

is printed as

```
[1, [2, [3, null]]]
```

- ***List notation***
  - Same as box notation, but any sub-structure that is a list is nicely formatted and printed as `list(...)`
  - Use predeclared function `display_list(x)` to show `x` in list notation

- **Example:**

```
display_list(
    pair(pair(pair(7, 8), pair(1, pair(2, null))),
         6));
```

prints

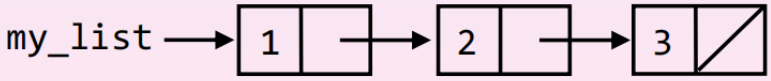
```
[list([7, 8], 1, 2), 6]
```

Show
Playgr

Box-and-pointer diagrams are graphical representations of data structures made of pairs

Example:

```
const my_list = pair(1, pair(2, pair(3, null)));
```



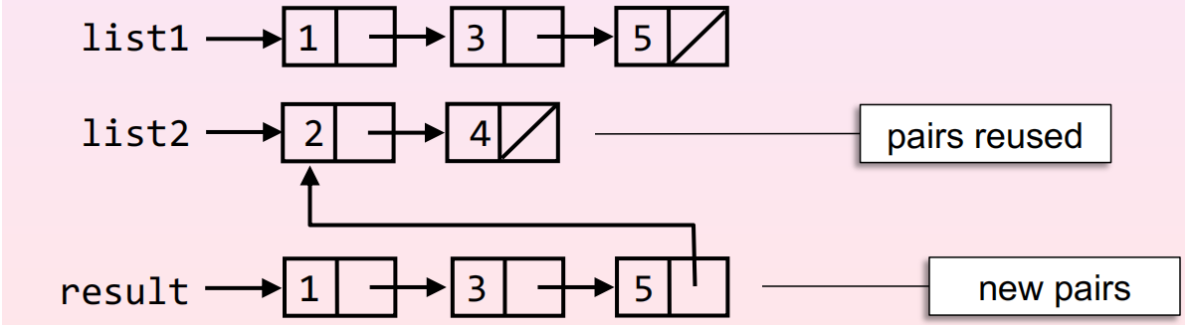
Data Visualizer tool generates such diagrams in Playground

- Use `draw_data` pre-declared function
- **Example:** `draw_data(pair(1, pair(2, pair(3, null))));`

```
//Length of Lists
function length(xs){
    return is_null(xs)
           ? 0
           : 1 + length(tail(xs));
}
function length_iter(xs){
    function len(ys,counted){
        return is_null(ys)
            ? counted
            : len(tail(ys),counted + 1);
    }
    return(xs,0);
}
```

```
//Appending Lists (Using Recursion)
function append(xs,ys){
    return is_null(xs)
        ? ys
        : pair(head(xs),append(tail,ys));
}
```

```
function append2(xs, ys) {
    return is_null(xs)
           ? ys
           : pair(head(xs), append2(tail(xs), ys));
}
const list1 = list(1, 3, 5);
const list2 = list(2, 4);
const result = append2(list1, list2);
```




```
//Reversing lists
function reverse(lst){
    return is_null(lst)
        ?null
        :append(reverse(tail(lst)),list(head(lst)))
}

function reverse(lst){
    function rev(original,reversed){
        return is_null(original)
            ?reversed
            :rev(tail(original),pair(head(original),reversed));
    }
    return rev(lst,null);
}
```

```
//Scaling lists (Enlarging every element by given ratio)
function scale_list(xs,k){
    return is_null(xs)
        ? null
        : pair(k*head(xs),scale_list(tail(xs),k));
}
//Squaring lists
function square_list(xs){
    const square = x => x*x;
    return is_null(xs)
        ?null
        :pair(square(head(xs)),square_list(tail(xs)));
}
```

```
//abstraction of given functions into lists
function map(f,xs){
    return is_null(xs)
```

```
        ? null
        : pair(f(head(xs)),map(f,tail(xs)));
}
function scale_list(xs,k){
    return map(x  => k*x,xs);
}
return square_list(xs){
    return map(x => x*x;xs);
}
function copy(xs){
    return map(x => x, xs);
}
```

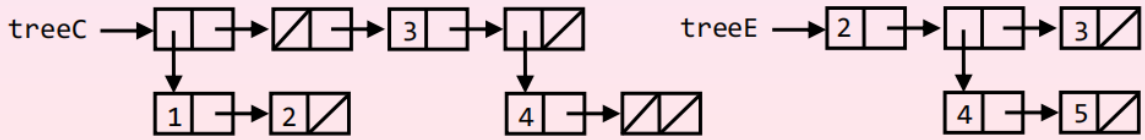
```
function even_numbers(xs){
    return is_null(xs)
        ? null
        :head(xs) % 2 ==== 0
        ?pair(head(xs),even_numbers(tail(xs)))
        :even_numbers(tail(xs));
}
```

```
//Adding elements in lists
function list_sum(xs){
    return is_null(xs)
        ? 0
        : head(xs) + list_sum(tail(xs));
}
function list_sum(xs){
    return accumulate((x,y) => x+y ,0,xs);
}
function accumulate(op, initial ,xs){
    return is_null(xs)
        ?initial
        :op(head(xs),accumulate(op,initial,tail(xs)));
}
function filter(pred,xs){
    return is_null(xs)
        ? null
        : pred(head(xs))
        ? pair(head(xs),filter(pred,tail(xs)))
        : filter(pred,tail(xs))
}
```

Tree

- A ***tree of a certain data type*** is
  - either **`null`**
  - or a **pair**
    - whose **tail is a tree of that data type** and
    - whose **head** is
      - either **of that data type**
      - or **a tree of that data type**

- Example trees of numbers:




```
function count_data_items(tree){
    return is_null(tree)
        ?0
        :(is_list(head(tree))
              ? count_data_items(head(tree))
              :1)+count_data_items(tail(tree));
}
```

```
//Scaling trees
function scale_tree(tree,k){
    return map(sub_tree =>
              ! is_list(sub_tree)
              ? k * sub_tree
              : scale_tree(sub_tree,k),tree);
}
function map_tree(f,tree){
    return map(sub_tree =>
              !is_list(sub_tree)
              ?f(sub_tree)
              :map_tree(f,sub_tree),tree);
}
//Changed version using map
function scale_tree(tree,k){
    return map_tree(data_item => data_item * k,tree);
}
```

Iterative append

- **First attempt:**

```
function append_iter(xs, ys) {
    return is_null(xs)
           ? ys
           : append_iter(tail(xs),
                         pair(head(xs), ys));
}

append_iter(list(1, 2, 3), list(4, 5, 6));
➔ list(3, 2, 1, 4, 5, 6)
```

- **Second attempt (using reverse):**

Show Playgr

```
function append_iter(xs, ys) {
    return is_null(xs)
           ? ys
           : append_iter(tail(xs), pair(head(xs), ys));
}
function append(xs, ys) {
    return append_iter(reverse(xs), ys);
}

append(list(1, 2, 3), list(4, 5, 6));
➔ list(1, 2, 3, 4, 5, 6)
```

```
function append(xs, ys) {  // Recursive process
    return is_null(xs)
           ? ys
           : pair(head(xs), append(tail(xs), ys));
}

function app(current_xs, ys, c) {  // Iterative process
    return is_null(current_xs)
           ? c(ys)
           : app(tail(current_xs), ys,
                 x => c(pair(head(current_xs), x)));
}
function append_iter(xs, ys) {
    return app(xs, ys, x => x);
}
```

Show in Playground

Continuation-Passing Style

Show in Playground

```
function app(current_xs, ys, c) {  // Iterative process
    return is_null(current_xs)
           ? c(ys)
           : app(tail(current_xs), ys,
                 x => c(pair(head(current_xs), x)));
}
function append_iter(xs, ys) {
    return app(xs, ys, x => x);
}
```

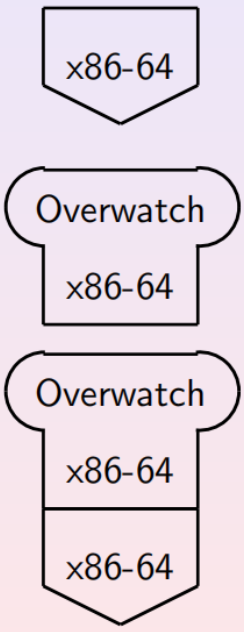
- **Programming Pattern: CPS**
  - Passing the deferred operation as a function in an extra argument is called "**Continuation-Passing Style**" (**CPS**)
  - We can convert ***any*** recursive function this way!

Sounds

(Source academy available)

Programming Language Processing

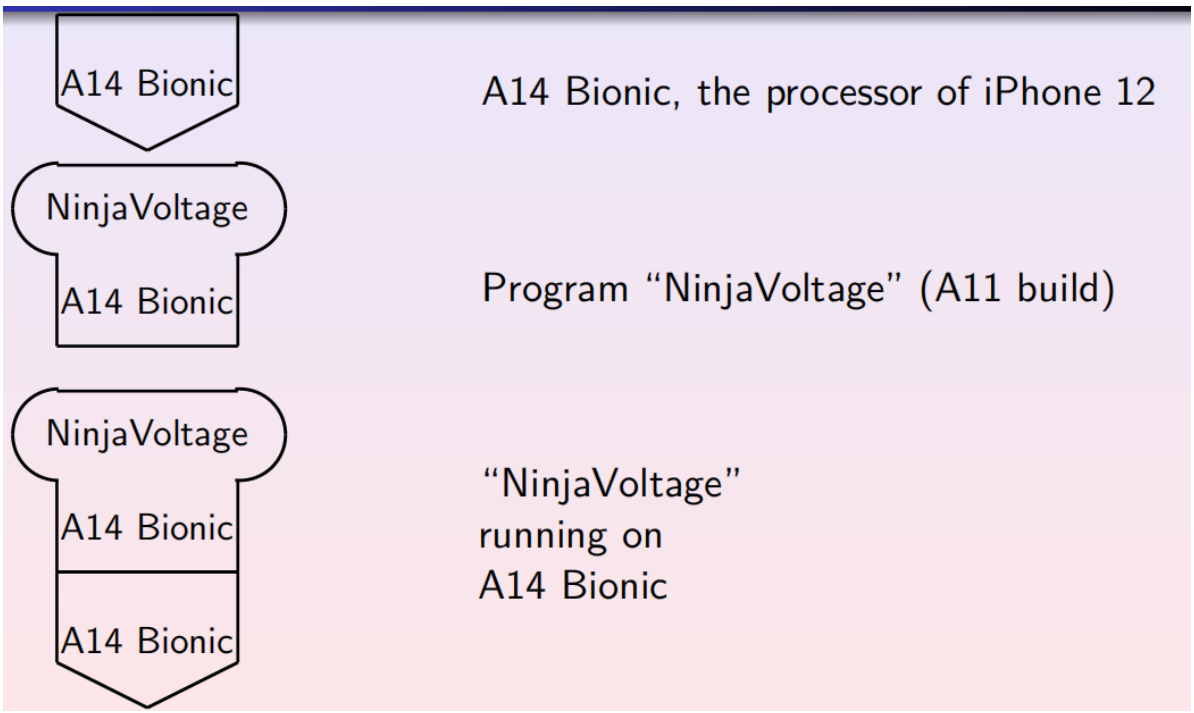
How programs run



x86-64 Processor

Program "Overwatch" (x86-64 code)

"Overwatch"
running on x86-64

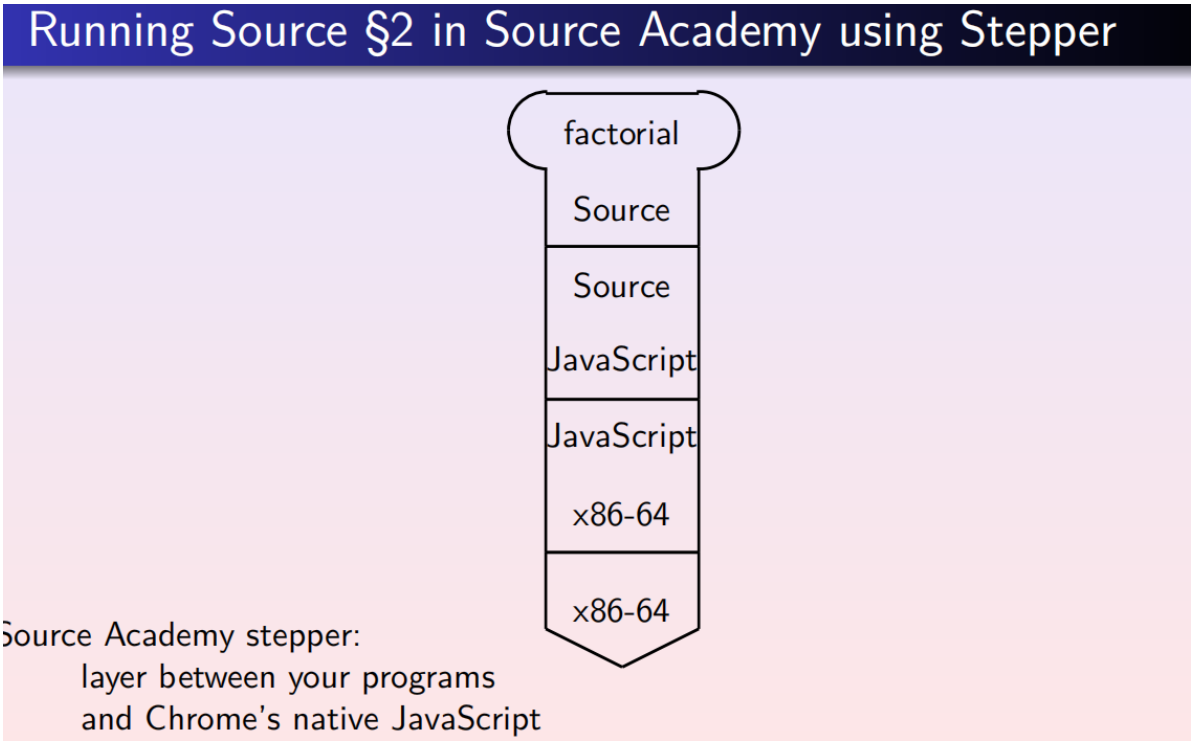


Interpreters

Interpreter is program that executes another program

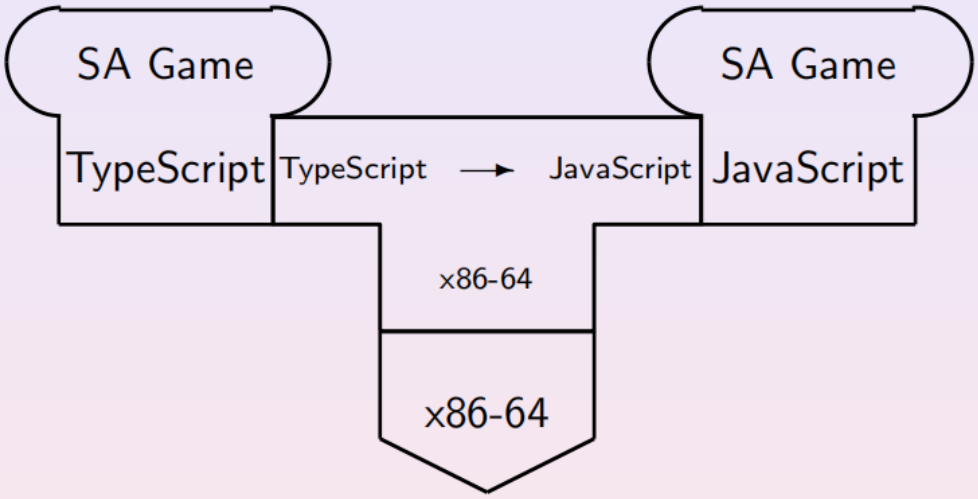
The interpreter's *source language* is the language in which the interpreter is written

The interpreter's *target language* is the language in which the programs are written which the interpreter can execute

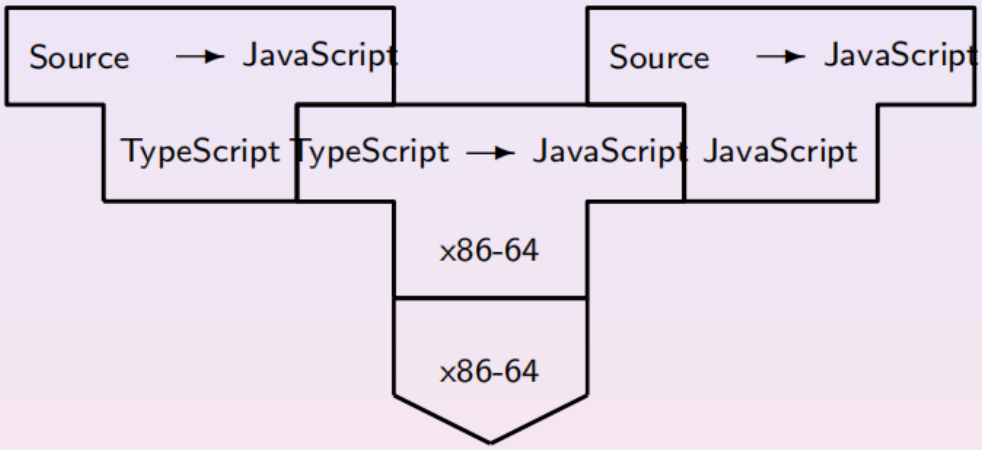


Compliers

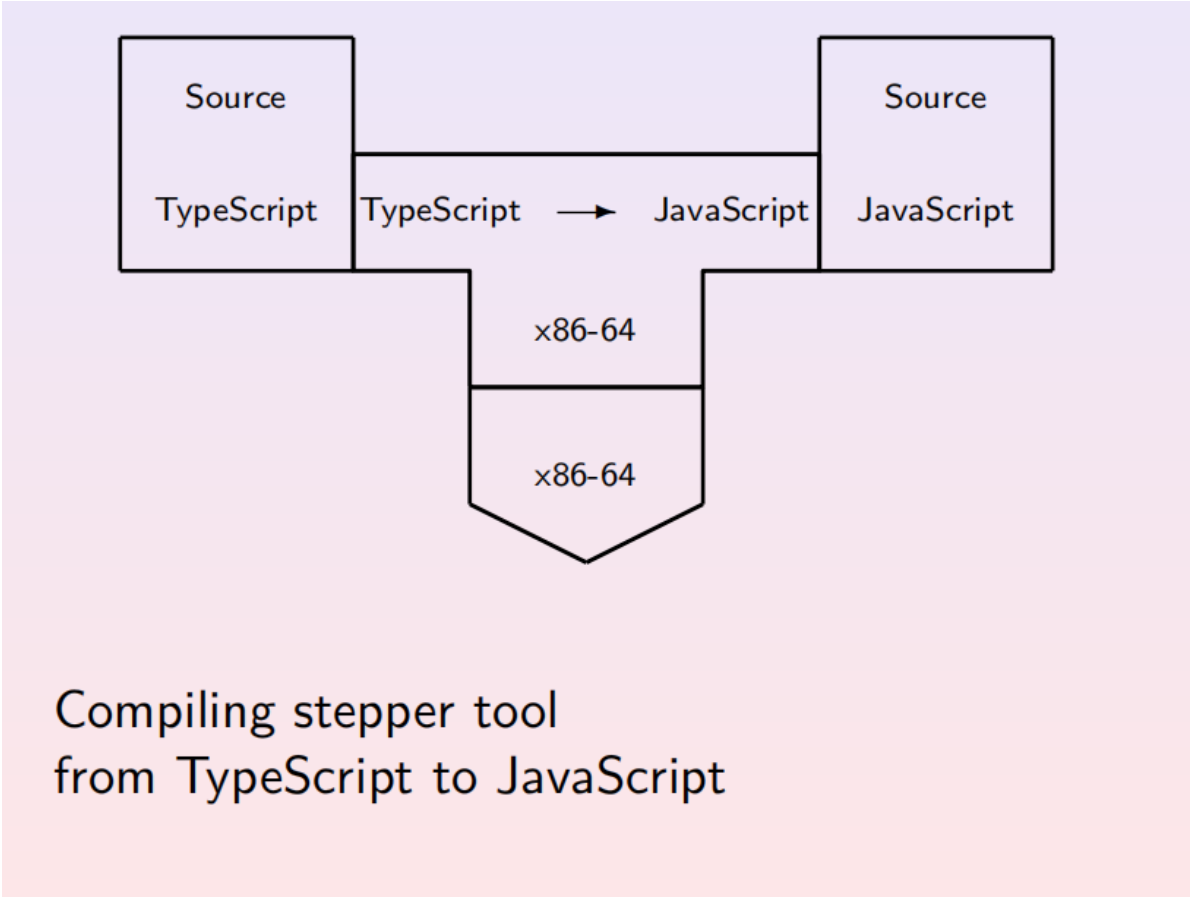
Definition: A compiler is a program that translates from one language (the *from-language*) to another language (the *to-language*)



Compiling “SA Game”
from TypeScript to JavaScript



Compiling
Source-to-JavaScript compiler
from TypeScript to JavaScript



Binary search & Symbolic processing