ES6+

ABSTRACT LOOP & LAZY EXECUTION

1 2 3 4 5

ABSTRACT LOOP

COMPLEX RECURSION

단순한 배열을 루프인 경우는 간단히 이터레이션을 작성할 수 있음.

COMPLEX RECURSION

단순한 배열을 루프인 경우는 간단히 이터레이션을 작성할 수 있음.

```
{
  [Symbol.iterator](){return this;},
   data:[1,2,3,4],
   next(){
      return {
         done:this.data.length == 0,
         value:this.data.shift()
      };
   }
}
```

COMPLEX RECURSION

복잡한 다층형 그래프는 어떻게 이터레이션할 것인가?

```
{
  [Symbol.iterator](){return this;},
   data:[{a:[1,2,3,4], b:'-'}, [5,6,7], 8, 9],
   next(){
      return ???;
   }
}
```

```
{
   [Symbol.iterator](){return this;},
   data:[{a:[1,2,3,4], b:'-'}, [5,6,7], 8, 9],
   next (){
      let v;
      while(v = this.data.shift()){
         switch(true){
         case Array.isArray(v):
            this.data.unshift(...v);
            break;
         case v && typeof v == 'object':
            for(var k in v) this.data.unshift(v[k]);
            break;
         default:
            return {value:v, done:false};
         }
      }
      return {done:true};
   }
}
```

```
{
   [Symbol.iterator](){return this;},
   data:[{a:[1,2,3,4], b:'-'}, [5,6,7], 8, 9],
   next(){
      let v;
      while(v = this.data.shift()){
         if(!(v instanceof Object)) return {value:v};
         if(!Array.isArray(v)) v = Object.values(v);
         this.data.unshift(...v);
      }
      return {done:true};
   }
}
```

```
const Compx = class{
   constructor(data){this.data = data;}
   [Symbol.iterator](){
     const data = JSON.parse(JSON.stringify(this.data));
     return {
       next(){
         let v;
         while(v = data.shift()){
           if(!(v instanceof Object)) return {value:v};
           if(!Array.isArray(v)) v = Object.values(v);
           data.unshift(...v);
         }
         return {done:true};
       }
     };
   }
};
const a = new Compx([{a:[1,2,3,4], b:'-'}, [5,6,7], 8, 9]);
console.log([...a]);
console.log([...a]);
```

```
const Compx = class{
   constructor(data){this.data = data;}
   *gene(){
     const data = JSON.parse(JSON.stringify(this.data));
     let v;
     while(v = data.shift()){
       if(!(v instanceof Object)) yield v;
       else{
         if(!Array.isArray(v)) v = Object.values(v);
         data.unshift(...v);
       }
     }
   }
};
const a = new Compx([{a:[1,2,3,4], b:'-'}, [5,6,7], 8, 9]);
console.log([...a.gene()]);
console.log([...a.gene()]);
```

ABSTRACT LOOP

다양한 구조의 루프와 무관하게 해당 값이나 상황의 개입만 하고 싶은 경우

ABSTRACT LOOP

다양한 구조의 루프와 무관하게 해당 값이나 상황의 개입만 하고 싶은 경우

```
(data, f)=>{
  let v;
  while(v = data.shift()){
    if(!(v instanceof Object)) f(v);
    else{
      if(!Array.isArray(v)) v = Object.values(v);
      data.unshift(...v);
    }
  }
}
```

ABSTRACT LOOP

다양한 구조의 루프와 무관하게 해당 값이나 상황의 개입만 하고 싶은 경우

```
(data, f)=>{
  let v;
  while(v = data.shift()){
    if(!(v instanceof Object)) f(v);
    else{
      if(!Array.isArray(v)) v = Object.
      data.unshift(...v);
    }
  }
}
```

```
(data, f)=>{
  let v;
  while(v = data.shift()){
    if(!(v instanceof Object)){
      console.log(v);
      f(v);
    }else{
      if(!Array.isArray(v)) v = Object.values(v);
      data.unshift(...v);
    }
  }
}
```

ABSTRACT LOOP

다양한 구조의 루프와 무관하게 해당 값이나 상황의 개입만 하고 싶은 경우

```
(data, f)=>{
  let v;
  while(v = data.shift()){
    if(!(v instanceof Object)) f(v);
    else{
      if(!Array.isArray(v)) v = Object.
      data.unshift(...v);
    }
  }
}
```

```
(data, f)=>{
  let v;
  while(v = data.shift()){
    if(!(v instanceof Object)){
      console.log(v);
      f(v);
    }else{
      if(!Array.isArray(v)) v = Object.values(v);
      data.unshift(...v);
    }
  }
}
```

제어문을 재활용할 수 없으므로
중복정의할 수 밖에 없다.

ABSTRACT LOOP

다양한 구조의 루프와 무관하게 해당 값이나 상황의 개입만 하고 싶은 경우

```
(data, f)=>{
  let v;
  while(v = data.shift()){
    if(!(v instanceof Object)){
      //v로 뭔가 하는 부분
    }else{
      if(!Array.isArray(v)) v = Object.values(v);
      data.unshift(...v);
    }
  }
}
```

ABSTRACT LOOP

결국 제어문을 직접 사용할 수 없고 구조객체를 이용해
루프실행기를 별도로 구현

ABSTRACT LOOP

결국 제어문을 직접 사용할 수 없고 구조객체를 이용해
루프실행기를 별도로 구현

ABSTRACT LOOP

결국 제어문을 직접 사용할 수 없고 구조객체를 이용해
루프실행기를 별도로 구현

```
(data, f)=>{
  let v;
  while(v = data.shift()){
    if(!(v instanceof Object)){
      f(v)
    }else{
      if(!Array.isArray(v)) v = Object.values(v);
      data.unshift(...v);
    }
  }
}
```

ABSTRACT LOOP

결국 제어문을 직접 사용할 수 없고 구조객체를 이용해
루프실행기를 별도로 구현

루프 공통 골격

```
(data, f)=>{
  let v;
  while(v = data.shift()){
    if(!(v instanceof Object)){
      f(v)
    }else{
      if(!Array.isArray(v)) v = Object.values(v);
      data.unshift(...v);
    }
  }
}
```

ABSTRACT LOOP

결국 제어문을 직접 사용할 수 없고 구조객체를 이용해
루프실행기를 별도로 구현

루프 공통 골격

```
(data, f)=>{
  let v;
  while(v = data.shift()){
    if(!(v instanceof Object)){
      f(v)
    }else{
      if(!Array.isArray(v)) v = Object.values(v);
      data.unshift(...v);
    }
  }
}
```

개별구조객체

ABSTRACT LOOP

팩토리 + 컴포지트

```
const Operator = class{
  static factory(v){
    if(v instanceof Object){
      if(!Array.isArray(v)) v = Object.values(v);
      return new ArrayOp(v.map(v=>Operator.factory(v)));
    }else return new PrimaOp(v);
  }
  constructor(v){this.v = v;}
  operation(f){throw 'override';}
};
const PrimaOp = class extends Operator{
  constructor(v){super(v);}
  operation(f){f(this.v);}
};
const ArrayOp = class extends Operator{
  constructor(v){super(v);}
  operation(f){for(const v of this.v) v.operation(f);}
};
Operator.factory([1,2,3,{a:4, b:5},6,7]).operation(console.log)
```

ABSTRACT LOOP

팩토리 + 컴포지트 + ES6 Iterable

```
const Operator = class{
  static factory(v){
    if(v instanceof Object){
      if(!Array.isArray(v)) v = Object.values(v);
      return new ArrayOp(v.map(v=>Operator.factory(v)));
    }else return new PrimaOp(v);
  }
  constructor(v){this.v = v;}
  *gene(){throw 'override';}
};
const PrimaOp = class extends Operator{
  constructor(v){super(v);}
  *gene(){yield this.v;}
};
const ArrayOp = class extends Operator{
  constructor(v){super(v);}
  *gene(){for(const v of this.v) yield * v.gene();}
};
for(const v of Operator.factory([1,2,3,{a:4, b:5},6,7]).gene()) console.log(v);
```

LAZY EXECUTION

YIELD

```
const odd = function*(data){
  for(const v of data){
    console.log("odd", odd.cnt++);
    if(v % 2) yield v;
  }
};
odd.cnt = 0;
for(const v of odd([1,2,3,4])) console.log(v);
```

YIELD

```
const odd = function*(data){
  for(const v of data){
    console.log("odd", odd.cnt++);
    if(v % 2) yield v;
  }
};
odd.cnt = 0;
for(const v of odd([1,2,3,4])) console.log(v);
```

```
const take = function*(data, n){
  for(const v of data){
    console.log("take", take.cnt++);
    if(n--) yield v; else break;
  }
};
take.cnt = 0;
for(const v of take([1,2,3,4], 2)) console.log(v);
```

YIELD

```
const odd = function*(data){
  for(const v of data){
    console.log("odd", odd.cnt++);
    if(v % 2) yield v;
  }
};
odd.cnt = 0;
for(const v of odd([1,2,3,4])) console.log(v);
```

```
const take = function*(data, n){
  for(const v of data){
    console.log("take", take.cnt++);
    if(n--) yield v; else break;
  }
};
take.cnt = 0;
for(const v of take([1,2,3,4], 2)) console.log(v);
```

```
for(const v of take(odd([1,2,3,4]), 2)) console.log(v);
```

YIELD*

```
const Stream = class{
  static get(v){return new Stream(v);}
  constructor(v){
    this.v = v;
    this.filters = [];
  }
  add(gene, ...arg){
    this.filters.push(v=>gene(v, ...arg));
    return this;
  }
  *gene(){
    let v = this.v;
    for(const f of this.filters) v = f(v);
    yield* v;
  }
};
```

YIELD*

```
const odd = function*(data){
  for(const v of data) if(v % 2) yield v;
};
const take = function*(data, n){
  for(const v of data) if(n--) yield v; else break;
};
for(const v of Stream.get([1,2,3,4]).add(odd).add(take, 2).gene())
console.log(v);
```