

Y-Combinator (不动点组合子)

```
const fact = (n) => n < 2 ? 1 : n * fact(n - 1);
```

将函数外提成为参数

The diagram illustrates a transformation in a lambda calculus expression. A horizontal line with a downward-pointing arrow on the left and a vertical line on the right connects the text above to the expression below. The expression is: `const P = fact => n => n < 2 ? 1 : n * fact(n - 1);`. In this expression, `const` and `fact` are blue, `P` is brown, `=` is black, `=>` is blue, `n` is blue, `<` is black, `2` is green, `?` is black, `1` is green, `:` is black, `n` is blue, `*` is black, `fact` is brown, `(` is blue, `-` is black, `1` is green, and `;` is black.

```
const P = fact => n => n < 2 ? 1 : n * fact(n - 1);
```

$$P(\text{fact}) = \text{fact};$$

fact即为函数映射P的**不动点**

Y-Combinator (不动点组合子)

```
const fact = (n) => n < 2 ? 1 : n * fact(n - 1);
```

将函数外提成为参数



```
const P = fact => n => n < 2 ? 1 : n * fact(n - 1);
```

```
P(fact) = fact;
```

fact即为函数映射P的不动点

如果有一个函数，能找到任意函数映射P的不动点
就能实现匿名函数的递归了

$Y(P) === \text{fact};$