

# ソースコードの注釈をプログラミングの知見として共有するソフトウェア

コーディング過程から学ぶプログラミング

## Disconomy

完成したソースコードから学ぶ「結果からの学習」ではなく、ソースコードが作られる道筋から学ぶ「過程からの学習」でプログラミング学習の負担を減らします。



### プログラムの解説

開発者がソースコードに解説を注釈付けることで意図や目的を伝えることができます。

### 過程の可視化

コーディング過程を見ることでソースコードがどのような順序で作られていったかが分かります。

### 知見の共有

Disconomyで作成した解説や過程はファイルに書き出して第三者に共有できます。

# テキストの注釈

テキストを使って論理的な説明ができます。

```

4  cgi = CGI.new
5  YOYFF
6  ファイル操作関数のgetsで読み取った文字列の末尾には改行文字が
7  付いているのでchomp関数で削除すべきである
8  ここは文字列の改行文字の有無なので直接的なエラーには関係ないかもしれな
9  いが
10  怪しい処理は早いうちに直しておくのが良い
11  ...while line = io.gets
12  ...  c += 1
13  ...  if c = 1 then
14  ...    puts "<h2>#{line}</h2>"
15  ...  else
16  ...    puts "<p><input type='checkbox' name=gumi value=#{line}>#{line}</p>"
17  ...  end
18  ...  io.flock(File::LOCK_UN)
19  ...  end
20  ...  print <<EOF
21  ...  <html>
22  ...  <body>
23  ...  <h1>投票システム</h1>
24  ...  <form action="http://cgi.u.tsukuba.ac.jp/~s2010392/wp/vote.rb" method="post">
25  ...  <p>
26  ...  <input type="submit" value="投票">
27  ...  <input type="reset" value="取り消し">
28  ...  </p>
29  ...  <a href="http://cgi.u.tsukuba.ac.jp/~s2010392/wp/view_result.rb">投票結果を見る</a>
30  ...  </body>
31  ...  </html>
32  ...  EOF
33  ...
34  ...
35  ...
36  ...

```

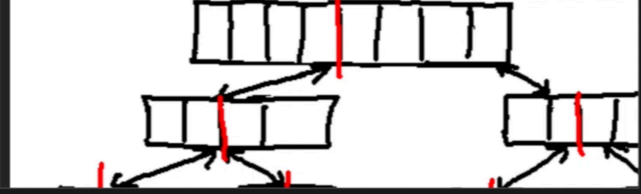
# 画像の注釈

画像を使って直感的な説明ができます。

```

1  void merge(int *a, int *tmp, int l, int r, int mid);
2  void merge_sort(int a[], int l, int r)
3  {
4  ... int tmp[r - l + 1];
5  ... int l;
6  ... YOYFF
7  ...
8  ... また言葉で説明するより画像で説明した方がいいときもあります
9  ...
10 ...
11 ...
12 ...
13 ...
14 ...
15 ...
16 ...
17 ... return;
18 ... }
19 ...
20 void merge(int *a, int *tmp, int l, int r, int mid)
21 {
22 ... int i;

```



# 注釈の装飾

装飾を使って視覚的な説明ができます。

```

1  void merge(int *a, int *tmp, int l, int r, int mid);
2  void merge_sort(int a[], int l, int r)
3  {
4  ... int tmp[r - l + 1];
5  ... int mid;
6  ... YOYFF
7  ...
8  ... Disconomyの注釈ではMarkdown記法が使えます
9  ... Boldにしたり、Italicにすることができます
10 ... これによって用語の強調が可能となり、学習者の理解を進めることができます
11 ...
12 ... merge_sort(a, l, mid);
13 ... merge_sort(a, mid + 1, r);
14 ...
15 ... merge(a, tmp, l, r, mid);
16 ...
17 ... return;
18 ... }
19 ...
20 void merge(int *a, int *tmp, int l, int r, int mid)
21 {
22 ... int i;

```

# 過程の可視化

コーディング過程の記録と注釈によるソースコードの解説ができます。

ソースコードがどのような順序で、どのような意図で作られていったのかを知ることができます。

「過程からの学習」によってプログラミング学習の負担を減らすことができます。

```

1  #!/usr/bin/env ruby
2  # encoding: utf-8
3  YOYFF
4  CGIのインスタンスを初期化
5  cgi = CGI.new
6

```

```

4  require 'cgi'
5
6  YOYFF
7  ファイルを文字コードUTF-8、読み取りモードで開く
8  その過程で例外が発生したら（例えばファイルが存在しない、ファイルを開くのに失敗したなど）
9
10 エラーメッセージを出力して処理を終了
11
12 rescue => ex
13 ... puts ex.message
14 ... exit
15 end

```

```

1  #!/usr/bin/env ruby
2  # encoding: utf-8
3
4  require 'cgi'
5
6  YOYFF
7  ファイルを文字コードUTF-8、読み取りモードで開く
8  その過程で例外が発生したら（例えばファイルが存在しない、ファイルを開くのに失敗したなど）
9
10 エラーメッセージを出力して処理を終了
11
12 rescue => ex
13 ... puts ex.message
14 ... exit
15 end

```

```

1  #!/usr/bin/env ruby
2  # encoding: utf-8
3
4  require 'cgi'
5
6  cgi = CGI.new
7
8  print cgi.header("text/html;charset=utf-8")
9
10 begin
11   f = open("question.txt", "r:UTF-8")
12 rescue => ex
13   puts ex.message
14   YOYFF
15   選択肢を格納する配列を初期化
16   selections = []
17

```