

# **Ruby 講義**

## **第6回 Ruby入門**

**Kuniaki IGARASHI/igaiga**

**2012.5.17 at 一橋大学**

**社会科学における情報技術とコンテンツ作成III  
(ニフティ株式会社寄附講義)**

○ 剰余金の配当に関するお知らせ

○ ニフティ、「@nifty EMOBILE LTE 定額にねんプラン」の提供を開...

○ 「@nifty温泉」で「母の日 全国一斉！100のありがとう風呂」特設サイト公...

○ 「スマブレ！」のサービス停止について

○ ニフティとサンリオウェブ、iOS向けアプリ「Hello Kitty Worl...

○ 平成24年3月期 決算短信

○ 特別損失の計上に関するお知らせ

○ 「シュフモ」登録会員数150万人を突破、「2012年 主婦の全国節電調査（冬季...

ニフティとなら、きっとかなう。  
With Us, You Can.

# ニフティ株式会社

HOME

NIFTY

ニュースリリース

アット・ニフティ

楽しいサービスがいっぱい

@nifty

アクセスマップ

大森から西新宿へ移転いたしました

@nifty Web募金

東日本大震災復興支援  
募金受付中



2012年4月25日 IR 特別損失の計上に関するお知らせ

2012年4月25日 IR 剰余金の配当に関するお知らせ

2012年4月25日 IR 平成24年3月期決算短信

2012年4月11日 ニフティ、「@nifty EMOBILE LTE 定額にねんプラン」の提供を開始

2012年4月11日 ニフティとサンリオウェブ、iOS向けアプリ「Hello Kitty World」を台湾で提供開始

2012年4月10日 お知らせ 「@nifty温泉」で「母の日 全国一斉！100のありがとう風呂」特設サイト公開



講師

五十嵐邦明

株式会社万葉

エンジニア



いりや  
@igatt

EM...  
編集後記とま

GARASHI

.9.25 at 高専カンファ



Teaching Assistant 濱崎 健吾  
クックパッド株式会社 エンジニア



先週の

おさらい

# 配列(Array)

教科書

p.33~

## ほかのオブジェクトの入れもの

作り方の例：

```
names = ["五十嵐", "濱崎"]
```

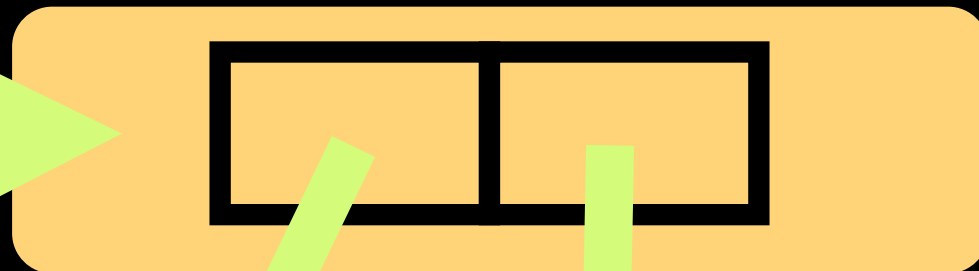
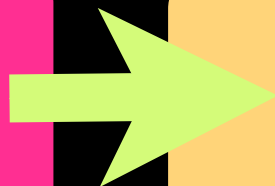
```
numbers = [1,3,5]
```

# 配列 (Array) 概念図

```
names = ["五十嵐", "濱崎"]
```

変数

● names



Array  
オブジェクト



五十嵐

濱崎

String  
オブジェクト

# 配列から読み込む

番号(index)を指定して読み込み

```
names = ["五十嵐", "濱崎"]
```

```
names[0] → "五十嵐"
```

```
names[1] → "濱崎"
```

最初の要素は0番です。1始まりではないので注意です。



# 配列へ追加する

配列の末尾にオブジェクトを追加するには  
pushメソッドを使います。

```
names = ["五十嵐", "濱崎"]
```

```
names.push("山田")
```

```
p names → ["五十嵐", "濱崎", "山田"]
```

# 配列の繰り返し処理

教科書

p.38~

**eachで中身を順番に処理する  
ものすごくーく大事！！！！**

```
配列.each do |変数|
```

```
  繰り返したい処理
```

```
end
```

# ハッシュ(Hash)

教科書

p.40~

## ほかのオブジェクトの入れもの

キーと値の組を持てます。なんでも入ります。

作り方の例：

```
recipe = {  
  :title => "♥日向夏のジャム♥",  
  :author => "濱崎" }
```

↑ キー

↑ 値



# オーソドックスなとんかつ ← title

レシピID : 1188379



揚げ物楽しい、豚肉安い、ソースも簡単

description ↑

author ↑

 hmskpad

豚ロース	1枚くらい
こしょう	少々
サラダ油	豚ロースが全て浸かるくらい

## ■ 衣

小麦粉（薄力粉）	100gくらい
卵	1個弱
パン粉	100gくらい

ingredients →

Hashを使う  
とこんな感じ  
でまとめられ  
ます

```
recipe = {  
  :title => "オーソドックスなとんかつ",  
  :author => "hmskpad",  
  :description => "揚げ物楽しい、豚肉安い、  
ソースも簡単",  
  :ingredients => [省略] }
```

# シンボル

教科書

p.41～

ラベルとして使う文字列的なもの

`:title`

シンボルにするには先頭に `:` を付ける

# ハッシュから読み込む

```
recipe = { :title => "♥日向夏のジャム♥",  
           :author => "濱崎" }  
p recipe[:title] → "♥日向夏のジャム♥"  
p recipe[:author] → "濱崎"
```

ハッシュ名[キー] で読み込みます。



# ハッシュへ追加する

ハッシュ名[キー] = 格納したいオブジェクト

同じキーの要素は追加不可(上書き)

ハッシュオブジェクト内でキーは唯一のもの(ユニーク)

```
recipe = { :title => "♥日向夏のジャム♥",  
  :author => "濱崎" }
```

```
recipe[:url] = "http://cookpad.com/recipe/xxx"
```

追加後

```
p recipe → { :title => "♥日向夏のジャム♥",  
  :author => "濱崎",  
  :url => "http://cookpad.com/recipe/xxx" }
```

# ハッシュの繰り返し処理

**Arrayと同じですが、変数を2個とります。**

```
ハッシュ.each do |キーの変数, 値の変数|  
  繰り返したい処理  
end
```

```
recipe = { :title => "♥日向夏のジャム♥", :author => "濱崎" }  
recipe.each do |k, v|  
  print k, " - ", v, "\n"  
end  
→ title - ♥日向夏のジャム♥  
   author - 濱崎
```

# ArrayとHashの使い分け

**Array : 順番が決まってるいれもの**

- 並び順が重要なもの
- データを重複させたい場合に利用

**Hash : キー(名札)を付けられるいれもの**

- 順番が保持されなくても困らないもの

※Ruby 1.9 からはHashも順番を保持します。

- キーが重複しない場合に利用



# nil

教科書

p.47

**「ない」ことを表すオブジェクト**

**例えば、ハッシュで存在しないキーを読もう  
とするとこの nil が返ってきます。**

**if 文などで条件判断をする場合、**

**nil は 偽（不成立）になります。**

**偽（不成立）になるのは false と nil の2つだけです。**

**それ以外の全ての値は真（成立）になります。**

今週

ここから

# 目次

## Wikipediaアクセス解析(前編)

2週間かけてWikipediaのアクセス数の解析の実習をします。

それに必要なRubyの知識を説明していきます。

データファイルから読み込み

while

unless

正規表現

例外処理



# Wikipediaのアクセス数解析

wikipediaは1時間ごとのアクセス数データを公開しています。

<http://dumps.wikimedia.org/other/pagecounts-raw/>

## Index of page view statistics for 2012-05

### Pagecount files for 2012-05

Check the [hashes](#) after your download, to make sure your files arrived intact.

- [pagecounts-20120501-000000.gz](#), size 69M
- [pagecounts-20120501-010000.gz](#), size 67M
- [pagecounts-20120501-020000.gz](#), size 67M
- [pagecounts-20120501-030000.gz](#), size 66M
- [pagecounts-20120501-040000.gz](#), size 67M
- [pagecounts-20120501-050000.gz](#), size 77M
- [pagecounts-20120501-060000.gz](#), size 75M
- [pagecounts-20120501-070000.gz](#), size 80M

# Wikipediaアクセス数データ

ja.b %C3%84 1 6499

ja.b %C3%88%C2%B1%C3%AF%C2%BF%C2%BD%C3%A7%C2%AC%C2%AC%C3%AF  
%C2%BD%C2%B3%C3%A6%C3%AF%C2%BF%C2%BD%C3%AF%C2%BF%C2%BD  
%C3%AF%C2%BD%C2%AC%C3%AF%C2%BD%C2%AC973%C3%A8%C2%AD%C3%AF  
%C2%BF%C2%BD%C3%AF%C2%BD%C2%A1 1 6656

ja.b %E3%81%95%E3%81%BE%E3%81%96%E3%81%BE%E3%81%AA%E9%9D  
%A2%E3%81%8B%E3%82%89%E8%A6%8B%E3%81%9F%E6%97%A5%E6%9C%AC\_  
%E5%9C%B0%E7%90%86\_%E6%B0%97%E5%80%99 1 18210

ja.b %E3%82%A6%E3%82%A3%E3%82%AD%E3%83%9A  
%E3%83%87%E3%82%A3%E3%82%A2%E3%81%AE%E6%9B%B8%E3%81%8D  
%E6%96%B9\_%E3%83%9D%E3%83%BC%E3%82%BF%E3%83%AB%E3%83%BB  
%E3%83%97%E3%83%AD%E3%82%B8%E3%82%A7%E3%82%AF  
%E3%83%88%E6%A1%88%E5%86%85 1 12093

ja.b %E3%82%A6%E3%82%A3%E3%82%AD%E3%83%9A  
%E3%83%87%E3%82%A3%E3%82%A2%E3%81%AE%E6%9B%B8%E3%81%8D  
%E6%96%B9\_%E5%85%A5%E9%96%80%E7%B7%A8-  
%E3%82%A6%E3%82%A3%E3%82%AD%E3%83%9A  
%E3%83%87%E3%82%A3%E3%82%A2%E3%81%A8%E3%81%AF%EF%BC%9F 1  
15837

...

っていうデータが数十万行

# Wikipediaアクセス数データ

データ構造を見るためにちょっと読みやすく変えたもの

ja.b アーティキュレーションを表す記号 1 7642

ja.b カテゴリ:スタブ 1 68732

ja.b カテゴリ:大学入試 3 45061

ja.b カテゴリ:民法 1 47619

ja.b カテゴリ:社会学 1 6661

ja.b カテゴリ:User\_bg 1 7931

ja.b カテゴリ:User\_uk-3 1 6599

ja.b ガス事業法第2条 1 8541

ja.b ガリア戦記 1 12936

ja.b ガリア戦記/参照画像一覧 1 54089

ja.b コントラクトブリッジ/ルール 2 7957

ja.b コントラクトブリッジ/ルール/スコアリング 1 14903

...

っていうデータが数十万行

# Wikipediaアクセス数解析

言語種別   ページタイトル   アクセス数   容量

ja.b %E3%81%84%E3%82%8D 1 6661

スペース区切りで以下の4項目が書かれている

言語種別： ja から始まるのが日本のデータ

ページタイトル： アクセスされたページ名

ただし、プログラムで扱い易い形式(%XX)になっている

アクセス数： アクセスされた回数

容量： そのページのデータサイズ

# Wikipediaアクセス数データ

言語種別   ページタイトル   アクセス数   容量

ja.b %C3%84 1 6499

ja.b %C3%88%C2%B1%C3%AF%C2%BF%C2%BD%C3%A7%C2%AC%C2%AC  
%C3%AF%C2%BD%C2%B3%C3%A6%C3%AF%C2%BF%C2%BD%C3%AF%C2%BF  
%C2%BD%C3%AF%C2%BD%C2%AC%C3%AF%C2%BD  
%C2%AC973%C3%A8%C2%AD%C3%AF%C2%BF%C2%BD%C3%AF%C2%BD  
%C2%A1 1 6656

ja.b %E3%81%95%E3%81%BE%E3%81%96%E3%81%BE%E3%81%AA%E9%9D  
%A2%E3%81%8B%E3%82%89%E8%A6%8B%E3%81%9F%E6%97%A5%E6%9C  
%AC\_%E5%9C%B0%E7%90%86\_%E6%B0%97%E5%80%99 1 18210

...

このデータを解析して、ある1時間のアクセス数トップ20をコードを書いて調べてみます。

簡単に言うと、「アクセス数」欄の数が大きいものから20個、その「ページタイトル」を表示させる

# Wikipediaのアクセス数解析

## コードで書く際の処理の流れ

- 🌀 データファイルを開く
- 🌀 データファイルから1行読み込む
- 🌀 日本語データ以外はパス
- 🌀 データ1行からタイトルとカウントを取得
- 🌀 取得データをいれものに詰めてとっておく
- 🌀 データファイル全行について繰り返し
- 🌀 データファイルを閉じる
- 🌀 貯まったデータをカウント順にソート（並べ替え）
- 🌀 トップ20件表示



# Wikipediaのアクセス数解析

```
# encoding: utf-8
```

```
require "cgi"
```

```
filename = "20120301-000000-ja.txt"
```

```
file = File.open(filename, "r:UTF-8")
```

```
list = []
```

```
while text = file.gets
```

```
  begin
```

```
    next unless text =~ /^ja/
```

```
    data = text.split
```

```
    h = {:title => CGI.unescape(data[1]), :count => data[-2]}
```

```
    list.push h
```

```
  rescue Exception => e
```

```
    p e
```

```
  end
```

```
end
```

```
file.close
```

```
# count順にソート
```

```
result = list.sort_by do |i|
```

```
  i[:count].to_i
```

```
end
```

```
# トップ20表示
```

```
result.reverse.first(20).each do |i|
```

```
  puts i
```

```
end
```

30行くらいで書けます

**各処理を  
解説していきます**

# Wikipediaのアクセス数解析

データファイルを開く

データファイルから1行読み込む

日本語データ以外はパス

データ1行からタイトルとカウントを取得

取得データをいれものに詰めてとっておく

データファイル全行について繰り返し

データファイルを閉じる

貯まったデータをカウント順にソート（並べ替え）

トップ20件表示

# Wikipediaのアクセス数解析

## コード解説

## データファイル名指定

```
require "cgi"
```

```
filename = "20120301-0000000-ja.txt"
```

```
file = File.open(filename, "r:UTF-8")
```

```
list = []
```

```
while text = file.gets
```

## 1行読み込み

## ファイル開く

(文字コードUTF-8)

```
begin
```

```
next unless text =~ /^ja/
```

```
data = text.split
```

```
h = {:title => CGI.unescape(data[1]), :count => data[-2]}
```

```
list.push h
```

## 各行がtextに

## 格納されて全行繰り返し

```
rescue Exception => e
```

```
pe
```

```
end
```

```
end
```

```
file.close
```

## ファイル閉じる

# ファイルから1行ずつ読み込み



教科書 p.51~55

## サンプルコード

```
filename = "20120301-0000000-ja.txt"
file = File.open(filename, "r:UTF-8")
while text = file.gets
  puts text
end
file.close
```

# ファイルを開く

```
filename = "20120301-0000000-ja.txt"
file = File.open(filename, "r:UTF-8")
while text = file.gets
  puts text
end
file.close
```

"**r:UTF-8**"

**r** は読み込みモード指定。  
readの意。

**UTF-8**は文字コード指定。

**File.open** メソッドでファイルを開きます。

(開く=データを読める状態にする)

開いたFileオブジェクトをfileへ代入しておきます。



# ファイルから各行読み込み

```
filename = "20120301-0000000-ja.txt"
file = File.open(filename, "r:UTF-8")
while text = file.gets
  puts text
end
file.close
```

**file.gets** : 1行読み込み

読み込めたらその行の内容を返す

読み込めなかったらnilを返す

**while ... end** : 条件が偽(false, nil)になるまで繰り返し

# while文, file.getsメソッド

```
filename = "20120301-000000-jar.txt"
file = File.open(filename)
while text = file.gets
  puts text
end
file.close
```

**while** **text = file.gets**

**while:** 条件成立中は繰り返し  
この場合、条件は  
**text = file.gets**  
になります。

**file.gets**

は1行読み込むメソッド  
読み込めたら真、  
ファイル終端で読み込めない  
と偽(nil)になります。



**while文は  
教科書 p.93**

# ファイルから各行読み込み

```
filename = "20120301-00000000-ja.txt"
file = File.open(filename, "r", :UTF-8)

while text = file.gets
  puts text
end
```

**while:** 条件成立中は繰り返し  
この場合、条件は  
**text = file.gets**  
になります。

**text** に1行目のデータが入って**end**まで処理

**while** の行へ戻り、

**text** に2行目のデータが入って**end**まで処理

... 全行繰り返し

終端で**file.gets**すると条件不成立になり繰り返し終了

# ファイルを閉じる

```
filename = "20120301-0000000-ja.txt"
file = File.open(filename, "r:UTF-8")
while text = file.gets
  puts text
end
file.close
```

**close** メソッドでファイルを閉じます。  
(閉じる=もう使わないよとRubyへ教えてあげる)

# Wikipediaのアクセス数解析

## データファイル名指定

```
require "cgi"
```

```
filename = "pagecounts-20120301-0000000"
```

```
file = File.open(filename, "r:UTF-8")
```

```
list = []
```

```
while text = file.gets
```

```
begin
```

```
  next unless text =~ /^ja/
```

```
  data = text.split
```

```
  h = {:title => CGI.unescape(data[1]), :count => data[-2]}
```

```
  list.push h
```

```
rescue Exception => e
```

```
  p e
```

```
end
```

```
end
```

```
file.close
```

ファイル開く

(文字コードUTF-8)

1行読み込み

各行がtextに

格納されて全行繰り返し

ファイル閉じる

# ファイル読み込みの演習

1. データファイル data.txt を作成して、  
data.txtの全行を表示する以下のコードを実行してください。

```
filename = "data.txt"  
file = File.open(filename, "r:UTF-8")  
while text = file.gets  
  puts text  
end  
file.close
```

Alice  
Bob  
Carol

↑ data.txtの例  
なんでもOKです。



# Wikipediaのアクセス数解析

データファイルを開く

データファイルから1行読み込む

日本語データ以外はパス

データ1行からタイトルとカウントを取得

取得データをいれものに詰めてとっておく

データファイルを閉じる

貯まったデータをカウント順にソート（並べ替え）

トップ20件表示

# Wikipediaのアクセス数解析

## コード解説

```
require "cgi"
filename = "pagecounts-20120301-000000"
file = File.open(filename, "r:UTF-8")
```

**ja.b %E3%81%84%E3%82%8D 1 6661**

```
list = []
```

```
while text = file.gets
  begin
```

日本語のデータは行先頭がja  
行先頭が ja でなかったらパス

**next unless text =~ /^ja/**

```
  data = text.split
```

```
  h = {:title => CGI.unescape(data[1]), :count => data[-2]}
```

```
  list.push h
```

```
rescue Exception => e
```

```
  p e
```

```
end
```

```
end
```

```
file.close
```

**next** : 繰り返し中の処理を次のデータに進める

**unless** : = if not

**text =~ /^ja/** : 正規表現

# next文



教科書 p.102

```
while text = file.gets
  ... (A) ...
  next
  ... (B) ...
end
```

繰り返し中にnext文を書くと、それ以降の処理を行わず、  
繰り返し先頭に戻します。

... (A) ... の部分の処理が行われてnextへくると、  
... (B) ... の部分の処理は行われず、  
繰り返し先頭へ戻ります。

教科書p.102の図6.2が分かりやすいです。

# unless文



教科書 p.76

```
unless 条件  
  処理  
end
```

条件が偽の時に処理を実行

if文と対になる文

if文とは逆に条件が偽の時に処理実行

以下の2つの文は同じ意味

```
puts "hello" if x != 3
```

```
puts "hello" unless x == 3
```

# 正規表現



教科書 p.44~46  
p.267~290

## 文字列がパターンに一致（マッチ）するか調べる道具

ものすごく便利で、強力で、奥が深いです。  
ここではごく基本的な説明だけを行います。

### 文字列 `=~` /正規表現パターン/

`=~` → 正規表現マッチを行う演算子

左辺の文字列中に正規表現パターンが含まれるかを判定します。  
パターンが英数字や漢字だけからなる場合、  
単純に文字列にパターンが含まれるかどうかを判定します。

```
"Ruby" =~ /Ruby/ → 0
```

# マッチした場合はマッチ位置を返します

```
"Ruby" =~ /Diamond/ → nil
```

# マッチしない場合はnilを返します

# 正規表現



教科書 p.44~46  
p.267~290

ほかにも便利な検索の機能があります。

```
text =~ /^ja/
```

正規表現パターン中の ^ は行先頭の意味

`/^ja/` は 対象文字列が `ja` から始まればマッチするパターンです。

```
"ja title count" =~ /^ja/ → 0
```

# 行先頭が `ja` から始まるのでマッチ

```
"xxxja" =~ /^ja/ → nil
```

# `ja`はあるが、行先頭ではないのでマッチしない



# Wikipediaのアクセス数解析

**next unless text =~ /^ja/**

**unless** は条件 **text =~ /^ja/** が不成立だったらその前の部分(**next**)を実行。

条件 **/^ja/** は正規表現でjaから始まる の意味。

**next**は以降の処理を行わずに次の繰り返しへ進む。

→**text** が **ja** から始まらなかった場合、  
以降の処理は行わず、  
次の繰り返し（次の行）へ進む。

# Wikipediaのアクセス数解析

## コード解説

```
while text = file.gets  
  begin
```

```
    next unless text =~ /^ja/
```

```
    data = text.split
```

```
    h = {:title => CGI.unescape(data[1]), :count => data[-2]}
```

```
    list.push h
```

```
  rescue Exception => e
```

```
    p e
```

```
  end
```

```
end
```

行先頭が ja でなかったら  
whileループの次の処理へ  
水色の部分は飛ばされる

# next, unless, 正規表現演習

1. データファイル data.txt を作成して、data.txtの全行のうち、行頭がBの行だけを表示する以下のコードを実行してください。

```
filename = "data.txt"
file = File.open(filename, "r:UTF-8")
while text = file.gets
  next unless text =~ /^B/
  puts text
end
file.close
```

Alice  
Bob  
Carol

↑ data.txtの例  
なんでもOKです  
が、B始まりの行  
と、そうでない行  
をいれてください

# Wikipediaのアクセス数解析

データファイルを開く

データファイルから1行読み込む

日本語データ以外はパス

データ1行からタイトルとカウントを取得

取得データをいれものに詰めてとっておく

データファイルを閉じる

貯まったデータをカウント順にソート（並べ替え）

トップ20件表示

# Wikipediaのアクセス数解析

データ1行からタイトルとカウントを取得

水色のところでやっています。

```
list = []
while text = file.gets
  begin
    next unless text =~ /^ja/
    data = text.split
    h = {:title => CGI.unescape(data[1]), :count
=> data[-2]}
    list.push h
  rescue Exception => e
    p e
  end
end
```

# String#split メソッド

**split**は文字列を空白文字で区切ってArrayにします。

```
data = list.split
```

```
"ja.b %E3%81%84%E3%82%8D 1 6661".split  
→ ["ja.b", "%E3%81%84%E3%82%8D", "1", "6661"]
```

# CGI.unescape メソッド

**%XX%XX**っていう謎の文字列を、読める形式に変換します。

```
require "cgi"
```

```
CGI.unescape("%E3%81%84%E3%82%8D%E3%81%AF")
```

```
→"いるは"
```

**この変換をURLデコードと呼びます。  
今はおまじないだと思ってください。**

# Wikipediaのアクセス数解析

```
data = text.split  
h = {:title => CGI.unescape(data[1]),  
     :count => data[-2]}
```

```
text→"ja.b %E3%81%84%E3%82%8D 1 6661"  
data→["ja.b", "%E3%81%84%E3%82%8D", "1", "6661"]  
h→{:title => "いる", :count => 1}
```

とある1行のタイトルとアクセス数が解析できました



# Wikipediaのアクセス数解析

データ1行からタイトルとカウントを取得

水色のところでやっています。

```
list = []
while text = file.gets
  begin
    next unless text =~ /^ja/
    data = text.split
    h = {:title => CGI.unescape(data[1]), :count
=> data[-2]}
    list.push h
  rescue Exception => e
    p e
  end
end
```

# Wikipediaのアクセス数解析

データファイルを開く

データファイルから1行読み込む

日本語データ以外はパス

データ1行からタイトルとカウントを取得

取得データをいれものに詰めてとっておく

データファイルを閉じる

貯まったデータをカウント順にソート（並べ替え）

トップ20件表示

# Wikipediaのアクセス数解析

データファイルを開く

データファイルから1行読み込む

日本語データ以外はパス

データ1行からタイトルとカウントを取得

**取得データをいれものに詰めてとっておく**

データファイルを閉じる

貯まったデータをカウント順にソート（並べ替え）

トップ20件表示

# Wikipediaのアクセス数解析

各行の整形データをArrayへ詰める

黄色のところでやっています。

```
list = []
```

```
while text = file.gets  
  begin
```

```
    next unless text =~ /^ja/
```

```
    data = text.split
```

```
    h = {:title => CGI.unescape(data[1]), :count => data[-2]}
```

```
    list.push h
```

```
  rescue Exception => e
```

```
    p e
```

```
  end
```

```
end
```

```
list = []
```

→list に空のArrayを代入

```
list.push h
```

list(Arrayオブジェクト)

の最後に hを追加

# Wikipediaのアクセス数解析

最後に、謎の構文を解説します。

桃色のところ。

```
list = []
```

```
while text = file.gets
```

```
  begin
```

```
    next unless text =~ /^ja/
```

```
    data = text.split
```

```
    h = {:title => CGI.unescape(data[1]), :count => data[-2]}
```

```
    list.push h
```

```
  rescue Exception => e
```

```
    p e
```

```
  end
```

```
end
```

# 例外処理



教科書

p.153~168

**begin**

例外が発生させる可能性のある処理

**rescue Exception => 変数**

例外が起こった場合の処理

**end**

コード実行中、うまく処理できない場合などに例外が発生させるメソッドがあります。例外が発生した場合、**rescue** 節で例外を捕まえ、その際に実行する特殊処理を書くことができます。

もしも、例外を**rescue** で捕まえない場合は、プログラムはそこでエラー終了します。

# 例外処理

```
list = []  
while text = file.gets  
  begin  
    next unless text =~ /^ja/  
    data = text.split  
    h = {:title => CGI.unescape(data[1]), :count => data[-2]}  
    list.push h  
    rescue Exception => e  
      p e  
    end  
  end  
end
```

具体的には、CGI.unescapeが変換できない状況になる場合に例外が発生します。例外を rescue 節で捕まえて、例外の内容 (e) をp メソッドで画面に表示するというコードです。例外については講義でまた説明する機会が出てくると思います。

# 演習

**1. データファイル data2.txt を作成して、data2.txt の全行を解析する以下のコードを実行してください。**

```
filename = "data2.txt"
file = File.open(filename, "r:UTF-8")
list = []
while text = file.gets
  data = text.split
  h = {:title => data[1], :count => data[-2]}
  list.push h
end
file.close
p list
```

```
ja Alice 2 100
ja Bob 1 200
ja Carol 3 300
```

←data2.txt



まとめ

# Wikipediaのアクセス数解析

## コードで書く際の処理の流れ

- 👁 データファイルを開く
- 👁 データファイルから1行読み込む
- 👁 日本語データ以外はパス
- 👁 データ1行からタイトルとカウントを取得
- 👁 取得データをいれものに詰めてとっておく
- 👁 データファイル全行について繰り返し
- 👁 データファイルを閉じる
- 👁 貯まったデータをカウント順にソート（並べ替え）
- 👁 トップ20件表示

ここまでできました。

# Wikipediaのアクセス数解析

```
# encoding: utf-8
```

```
require "cgi"
```

```
filename = "20120301-000000-ja.txt"
```

```
file = File.open(filename, "r:UTF-8")
```

```
list = []
```

```
while text = file.gets
```

```
  begin
```

```
    next unless text =~ /^ja/
```

```
    data = text.split
```

```
    h = {:title => CGI.unescape(data[1]), :count => data[-2]}
```

```
    list.push h
```

```
  rescue Exception => e
```

```
    p e
```

```
  end
```

```
end
```

```
file.close
```

```
# count順にソート
```

```
result = list.sort_by do |i|
```

```
  i[:count].to_i
```

```
end
```

```
# トップ20表示
```

```
result.reverse.first(20).each do |i|
```

```
  puts i
```

```
end
```

ここまでできました。

# ファイルから1行ずつ読み込み



教科書 p.51~55

## サンプルコード

```
filename = "20120301-0000000-ja.txt"
file = File.open(filename, "r:UTF-8")
while text = file.gets
  puts text
end
file.close
```

# ファイルから各行読み込み

```
filename = "20120301-00000000-ja.txt"
file = File.open(filename, "r", :UTF-8)

while text = file.gets
  puts text
end
```

**while:** 条件成立中は繰り返し  
この場合、条件は  
**text = file.gets**  
になります。

**text** に1行目のデータが入って**end**まで処理

**while** の行へ戻り、

**text** に2行目のデータが入って**end**まで処理

... 全行繰り返し

終端で**file.gets**すると条件不成立になり繰り返し終了

は1行読み込むメソッド

読み込むメソッド

ファイル終端で読み込めない

条件不成立になり繰り返し終了

# next文



教科書 p.102

```
while text = file.gets
  ... (A) ...
  next
  ... (B) ...
end
```

繰り返し中にnext文を書くと、それ以降の処理を行わず、繰り返し先頭に戻します。

... (A) ... の部分の処理が行われてnextへくると、  
... (B) ... の部分の処理は行われず、  
繰り返し先頭へ戻ります。

教科書p.102の図6.2が分かりやすいです。

# unless文



教科書 p.76

```
unless 条件  
  処理  
end
```

条件が偽の時に処理を実行

if文と対になる文

if文とは逆に条件が偽の時に処理実行

以下の2つの文は同じ意味

```
puts "hello" if x != 3
```

```
puts "hello" unless x == 3
```

# 正規表現



教科書 p.44~46  
p.267~290

## 文字列がパターンに一致（マッチ）するか調べる道具

ものすごく便利で、強力で、奥が深いです。  
ここではごく基本的な説明だけを行います。

### 文字列 `=~` /正規表現パターン/

`=~` → 正規表現マッチを行う演算子

左辺の文字列中に正規表現パターンが含まれるかを判定します。  
パターンが英数字や漢字だけからなる場合、  
単純に文字列にパターンが含まれるかどうかを判定します。

```
"Ruby" =~ /Ruby/ → 0
```

# マッチした場合はマッチ位置を返します

```
"Ruby" =~ /Diamond/ → nil
```

# マッチしない場合はnilを返します



# Wikipediaのアクセス数解析

**next unless text =~ /^ja/**

**unless** は条件 **text =~ /^ja/** が不成立だったらその前の部分(**next**)を実行。

条件 **/^ja/** は正規表現でjaから始まる の意味。

**next**は以降の処理を行わずに次の繰り返しへ進む。

→**text** が **ja** から始まらなかった場合、  
以降の処理は行わず、  
次の繰り返し（次の行）へ進む。

# String#split メソッド

**split**は文字列を空白文字で区切ってArrayにします。

```
data = list.split
```

```
"ja.b %E3%81%84%E3%82%8D 1 6661".split  
→ ["ja.b", "%E3%81%84%E3%82%8D", "1", "6661"]
```

# CGI.unescape メソッド

**%XX%XX**っていう謎の文字列を、読める形式に変換します。

```
require "cgi"
```

```
CGI.unescape("%E3%81%84%E3%82%8D%E3%81%AF")
```

```
→"いるは"
```

**この変換をURLデコードと呼びます。  
今はおまじないだと思ってください。**

# 例外処理



教科書

p.153~168

**begin**

**例外が発生させる可能性のある処理**

**rescue Exception => 変数**

**例外が起こった場合の処理**

**end**

コード実行中、うまく処理できない場合などに例外が発生させるメソッドがあります。例外が発生した場合、**rescue** 節で例外を捕まえ、その際に実行する特殊処理を書くことができます。

もしも、例外を**rescue** で捕まえない場合は、プログラムはそこでエラー終了します。



# 参考文献

**ウィキペディア記事閲覧回数の特徴分析**

**人工知能学会研究会資料 SIG-SWO-A901-03**

**曽根 広哲、山名 早人**

**早稲田大学大学院基幹理工学研究科 早稲田大学理工学術院 国立情報学研究所**

**<http://sigswo.org/papers/SIG-SWO-A901/SIG-SWO-A901-03.pdf>**

# 講義資料置き場

講義資料置き場をつくりました。  
過去の資料がDLできます。

<https://github.com/hitotsubashi-ruby/lecture2012>  
or

<http://bit.ly/ruby-lecture>

# 雑談・質問用facebookグループ

facebookグループを作りました

<https://www.facebook.com/groups/hitotsubashi.rb>

- 加入/非加入は自由です
- 加入/非加入は成績に関係しません
- 参加者一覧は公開されます
- 書き込みは参加者のみ見えます
- 希望者はアクセスして参加申請してください
- 雑談、質問、議論など何でも気にせずどうぞ～
- 質問に答えられる人は答えてあげてください
- 講師陣もお答えします
- 入ったら軽く自己紹介おねがいします