基礎プログラミングII

第9回　チーム課題準備 (1)

メディア情報コース
平居　悠（ひらい　ゆたか）

到達目標

プログラミングを用いた実践的なデータ処理と情報システムの構築

- 定式化された処理を**関数の形**で記述し利用することができるようになる
- 再帰などの**アルゴリズム**を理解し問題に適用できるようになる
- 基礎的な**CGI**の仕組みの理解を通して**Webインターフェース**を設計できるようになる
- 多様な**社会事象への適用**を設計できるようになる
- 現実社会の課題に対応する**情報システム**を設計・作成できるようになる
- **生成AI**を学習の道具として利用できるようになる

前回

第１回　生成AIを効果的に利用した学習法
第２回　メソッド定義と効果的活用（関数的処理）
第３回　メソッド定義と効果的活用（データ集合処理）
第４回　メソッド定義と効果的活用（再帰的アルゴリズム）
第５回　変数のスコープ・クラス設計
第６回　専門演習紹介
第７回　CGIと情報システム（1）
第８回　CGIと情報システム（2）
第９回　チーム課題準備期間（1）
第１０回　チーム課題準備期間（2）・trr試験
第１１回　予選発表
第１２回　代表発表
第１３回　合同成果発表会
第１４回　期末試験

前回の目標

LaTeXを用いて簡単な文書を作成できるようになる。

LaTeX（ラテフ）

論文やレポート記事などを出版物と同じ品質で作成されるために設計されたマークアップ言語

LaTeXによる文書作成手順

1. 本文（ソース）作成
2. 組み版処理（タイプセット）
3. 印刷

LaTeXソースの構造

```
\documentclass[uplatex]{jsarticle}
\begin{document}
こんにちは!
\end{document}
```

jsarticle: 文書スタイル (他にjsbookやjsreportなど)
document環境：この中に本文を記述

LaTeXの文法

- LaTeX処理システムに特別に与える記号はバックスラッシュで始める
- バックスラッシュと英単語で始まるものを**マクロ**という
- マクロには{ }で括って引数を与えることがある
- \begin{}と\end{}に囲まれた部分を**環境**と呼ぶ。

document環境

\begin{document}, \end{document}で文書本体の開始と終わりを宣言する。
ファイルの先頭から\begin{document}までの部分を**プリアンブル**という。

```
\documentclass[uplatex]{jsarticle}
（プリアンブル）
\begin{document}
（文書本体）
\end{document}
```

前回の問い

- LaTeXとは何か？
  - 文書を作成するために設計されたマークアップ言語

タイピング練習スケジュール

第２回　　ホームポジション
第３回　　ローマ字
第４回　　英語初級
第５回　　日本国憲法
第６回　　ホームポジション
第７回　　ローマ字
第８回　　英語初級
第９回　　日本国憲法
第１０回　日本国憲法（trr試験、合格点：200点）

タイピングの練習 (日本国憲法)

1. ブラウザを起動し、https://www.koeki-prj.org/trr/ に繋ぐ。
2. 学籍番号（Cは大文字、省略なし８桁）を入力する。
3. Koeki MAILに届いたパスコードをPasscode: 欄に入力する。

今回

第１回　生成AIを効果的に利用した学習法
第２回　メソッド定義と効果的活用（関数的処理）
第３回　メソッド定義と効果的活用（データ集合処理）
第４回　メソッド定義と効果的活用（再帰的アルゴリズム）
第５回　変数のスコープ・クラス設計
第６回　専門演習紹介
第７回　CGIと情報システム（1）
第８回　CGIと情報システム（2）
第９回　チーム課題準備期間 (1)
第１０回　チーム課題準備期間 (2) ・trr試験
第１１回　予選発表
第１２回　代表発表
第１３回　合同成果発表会
第１４回　期末試験

今回の目標

CGIを用いてWebインターフェースを設計できるようになる。

今回学ぶこと

1. 検索主体のCGIプログラムの作成
2. 紙芝居のようにページを進めるCGI
3. データを永続させるCGI
4. CGIへの画像投稿
5. システム設計

今回の問い

- データベース検索をCGI化するにはどのようにすれば良いか？
- １つのCGIプログラムで紙芝居のように順次移り変わるWebページを生成するにはどうすれば良いのか？

今回学ぶこと

1. 検索主体のCGIプログラムの作成
2. 紙芝居のようにページを進めるCGI
3. データを永続させるCGI
4. CGIへの画像投稿
5. システム設計

コンソールプログラム

コマンドラインで起動し、入出力を
すべてコマンドラインで完了させる
プログラム

コンソールプログラム

コンソールプログラムで値を利用者から読み取るときは主にgetsを利用しその結果を変数に代入する。

```
STDERR.print(“年齢を入力してください：”)
age = gets.to_i
```

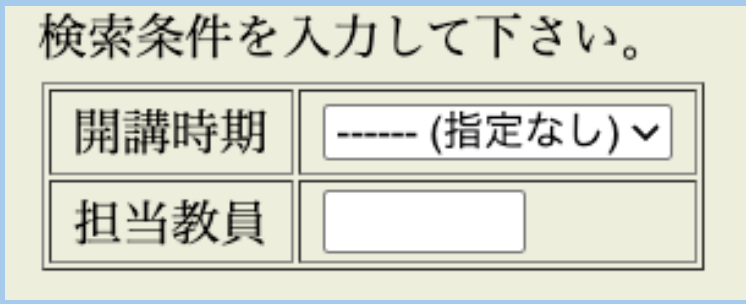
コンソールプログラムのCGI化

前ページのようなものをCGI化するには、入力に関わる部分をCGIパラメータから受け取るように変え、出力する部分をHTML形式で書き出すように変更する。

コンソールプログラムのCGI化

履修科目データベース検索をCGI化する。「開講時期」をjiki、
「担当教員」をkyoiで受け渡すHTMLフォームは以下の通り。

```
<form method="POST" action="db-cgi.rb">
 <p>検索条件を入力して下さい。</p>
 <table border="1">
  <tr><td>開講時期</td><td><select name="jiki">
                                  <option value="">------ (指定なし)</option>
                                  <option>春学期</option>
                                  <option>秋学期</option>
                                 </select></td></tr>
  <tr><td>担当教員</td><td><input name="kyoi" size="8"></td></tr>
 </table>
</form>
```



コンソールプログラムのCGI化

入力される値をjiki, kyoiから受け取るRubyプログラムは以下の通り。

```
require 'cgi'
c = CGI.new(:accept_charset => "UTF-8")
term = c["jiki"]
whom = c["kyoi"]
```

この結果出力をHTML形式で行えばCGI化は完了する。

search-cgi.rb 前半 （入力部分）

```
#!/usr/bin/env ruby
# coding: utf-8
Encoding.default_external = 'utf-8'     # UTF-8のCSVファイルを読むため
require 'cgi'
require 'csv'
c = CGI.new(:accept_charset => "UTF-8")

#データベース読み取り処理は全く同じ
csv = CSV.read("syllabus.csv", headers: true)
db  = csv               # あとで全データを使う場合にそなえcsv変数は温存

# 検索パターンはHTMLフォームへの入力値を取得する
term = c["jiki"]
whom = c["kyoi"]

if term > ""                                    # 開講時期指定に何か文字列を入れたなら
 ptn = Regexp.new(term)     # 文字列を正規表現に変換
 db = db.select {|row| ptn =~ row["開講時期"]}
end
if whom > ""                                    # 担当教員指定に何か文字列を入れたなら
 ptn = Regexp.new(whom)   # 文字列を正規表現に変換
 db = db.select {|row| ptn =~ row["担当教員"]}
end
```

search-cgi.rb後半（出力部分）

Rubyプログラムの“”内でHTMLの“”を使う際は、\“\”とする。

```
print("Content-type: text/html; charset=utf-8

<!DOCTYPE html>
<html>
<head><title>検索結果</title>
<style type=\"text/css\">
<!--
 tr.head {background: cyan;}
 th {width: 8em;}
 -->
</style>
</head>
<body>")
puts("<h1>該当科目一覧</h1>")
puts("<table border=\"1\">")
db.each {|row|
  printf("<tr class=\"head\"><td colspan=\"2\">%s</td></tr>\n",
       row["科目名"])
  row.each {|key, value|
    printf("<tr><th>%s</th><td>%s</td></tr>\n", key, value)
  }
}
puts("</table>")
puts("</body>
</html>")
```

入力フォームの自動出力

検索オプションが多数になるとoption要素一覧を用意するのが困難になる。

データの値に関連する選択肢を入力フォームにいれる場合、入力フォーム自体もデータの値を管理するCGIプログラムで生成すると良い。

search-cgi.rbで読み込むデータベース

値

ハッシュ

科目名	基礎プログラミング
科目コード	154
担当教員	鳥海三輔
開講時期	春学期
概要	プログラミングを通じて問題解決能力の向上を図る。

ハッシュ

科目名	応用プログラミング
科目コード	254
担当教員	飯森大和
開講時期	秋学期
概要	C言語を通じて問題解決に最適なアルゴリズムを適用する方策を学ぶ。

ハッシュ

科目名	応用もっけ論
科目コード	39
担当教員	酒田育造
開講時期	春学期
概要	その地域の「もっけ」を理解し、深い交流のきっかけとする。

search-cgi.rb中の変数から「開講時期」の値一覧を得る手順

db[“開講時期”]

→[“春学期”, “秋学期”, “春学期”]

db[“開講時期”].uniq

→[“春学期”, “秋学期”]

search-cgi.rb中の変数から「開講時期」の値一覧を得る手順

```
db[“開講時期”].uniq.collect {|j|
“<option>#{j}</option>” }.join(“\n”)
```

→

<option>春学期</option>
<option>秋学期</option>

form-search-cgi.rb前半

```
#!/usr/bin/env ruby
# coding: utf-8
Encoding.default_external = 'utf-8'
require 'csv'
require 'cgi'
c = CGI.new(:accept_charset => "UTF-8")

#データベース読み取り処理は全く同じ
db = CSV.read("syllabus.csv", headers: true)
opts = db["開講時期"].uniq.collect {|j| "<option>#{j}</option>"}.join("\n")

# HTMLヘッダと入力フォーム出力
print <<EOF
Content-type: text/html; charset=utf-8

<!DOCTYPE html>
<html>
<head><title>科目検索</title>
<style type="text/css">
<!--
 tr.head {background: cyan;}
 th {width: 8em;}
 -->
</style>
</head>
<body>
<form method="POST" action="form-search-cgi.rb">
 <h1>科目検索</h1>
 <p>検索条件を入力して下さい。</p>
 <table border="1">
  <tr><td>開講時期</td><td><select name="jiki">
                                        <option value="">------ (指定なし)</option>
EOF

print(opts)     # これで自動生成した option 一覧が出る
```

form-search-cgi.rb後半

```
print <<EOF
</select></td></tr>
 <tr><td>担当教員</td><td><input name="kyoi" size="8"></td></tr>
 </table>
 <p><input name="ok" type="submit" value="検索">
 <input name="cl" type="reset" value="リセット"></p>
</form>
EOF
# 検索パターンはHTMLフォームへの入力値を取得する
term = c["jiki"]
whom = c["kyoi"]

if term>"" or whom>""                    # 検索条件の最低1つが入力されていたら
 if term > ""                            # 開講時期指定に何か文字列を入れたなら
  ptn = Regexp.new(term)    # 文字列を正規表現に変換
  db = db.select {|row| ptn =~ row["開講時期"]}
 end
 if whom > ""                            # 担当教員指定に何か文字列を入れたなら
  ptn = Regexp.new(whom) # 文字列を正規表現に変換
  db = db.select {|row| ptn =~ row["担当教員"]}
 end

 puts("<h2>該当科目一覧</h2>")
 printf("<p>検索語: 開講時期=[%s],  担当教員=[%s]</p>", term, whom)
 puts("<table border=\"1\">")
 db.each {|row|
  printf("<tr class=\"head\"><td colspan=\"2\">%s</td></tr>\n",
       row["科目名"])
  row.each {|key, value|
   printf("<tr><th>%s</th><td>%s</td></tr>\n", key, value)
  }
 }
 puts("</table>")
end
puts("</body>\n</html>")
```

データベースからHTML文を出力する手順

1. データベースを読み込む
2. 自分自身を呼ぶform文を出力する
3. formの値が入力されていたら検索結果を出力

今回の問い

- データベース検索をCGI化するにはどのようにすれば良いか？
  - 入力に関わる部分をCGIパラメータから受け取り、出力する部分をHTML形式で書き出す。
- １つのCGIプログラムで紙芝居のように順次移り変わるWebページを生成するにはどうすれば良いのか？

今回学ぶこと

1. 検索主体のCGIプログラムの作成
2. **紙芝居のようにページを進めるCGI**
3. データを永続させるCGI
4. CGIへの画像投稿
5. システム設計

紙芝居のようにページを進めるCGI

1つのCGIプログラムで紙芝居のように順次移り変わるWebページを生成する。

hidden変数の活用

1つのCGIプログラムに複数のWebページ出力を管理させるには、現在の状態を示す変数を**hidden変数**に埋め込んで、次に起動されるCGIプログラムでその値を参照して出力するHTMLを切り替えればよい。

hidden変数の活用

以下の流れで進むWebページ遷移を考える。

1. こんにちは
2. やあやあ
3. さようなら

CGIプログラムの方針

1. 出力すべきページの指定がない場合
   →１ページ目を出力
2. 出力すべきページの指定がある場合
   →そのページを出力

CGIプログラムの方針

ページ番号とそのページでの出力内容
を以下のようなハッシュで定義する。

```
page = {
  "1" => "こんにちは",
  "2" => "やあやあ",
  "3" => "さようなら",
}
```

CGIプログラムの方針

出力すべきページ番号を入力名pageで管理するならHTMLで以下のフォームを記述すれば良い。

```
<form method="POST" action="page2go.rb">
 <input type="hidden" name="page" value="ページ番号">
</form>
```

page2go.rb前半

```
#!/usr/bin/env ruby
# coding: utf-8

require 'cgi'
c = CGI.new(:accept_charset => 'utf-8')

page = {
 "1" => "こんにちは",
 "2" => "やあやあ",
 "3" => "さようなら",
}
p = c["page"]
if p==""
 p="1"                                  # page変数指定がなければ "1" とする
end
name = c["name"]

puts "Content-type: text/html; charset=utf-8

<!DOCTYPE html>
<html>
<head><title>Page #{p}</title></head>
<body>
<h1>Page #{p}</h1>
<form method=\"POST\" action=\"page2go.rb\">"
```

page2go.rb後半

```
# ページ番号による切り替え処理
case p
when "1"
  nextpage = "2"
when "2"
  nextpage = "3"
when "3"
  nextpage = nil                          # 「次のページ」はなし
end
if name > ""                              # nameが定義されていたら呼びかけ文出力
  printf("<p>%sさん!</p>\n", name)
end
printf("<p>%s</p>\n", page[p])          # ページ番号に応じたメッセージ出力

if nextpage
  if name == ""                                    # 名前入力がなければ名前入力フォーム
    puts('<p>お名前: <input type="text" name="name" size="10"></p>')
  else
    # 既に入力されていたら hidden 変数に保存
    printf("<input type=\"hidden\" name=\"name\" value=\"%s\">\n", name)
  end
  printf("<input type=\"hidden\" name=\"page\" value=\"%s\">\n", nextpage)
  puts('<input type="submit" value="次へ">')
end

puts "</form>
</body>
</html>"
```

今回の問い

- データベース検索をCGI化するにはどのようにすれば良いか？
  - 入力に関わる部分をCGIパラメータから受け取り、出力する部分をHTML形式で書き出す。
- **１つのCGIプログラムで紙芝居のように順次移り変わるWebページを生成するにはどうすれば良いのか？**
  - **hidden変数を活用する。**

今回学ぶこと

1. 検索主体のCGIプログラムの作成
2. 紙芝居のようにページを進めるCGI
3. **データを永続させるCGI**
4. CGIへの画像投稿
5. システム設計

CGI変数の寿命

CGIでは、webページで入力された値はCGIスクリプトの終了とともに消えてしまう。

PStoreクラス

現在の変数（数値、文字列、配列、ハッシュなど）の値をそのままの形でファイルに保存できる。

PStoreクラスにハッシュの値を保存する例

```
require 'pstore'
x = PStore.new("値保存ファイル")
x.transaction do
  x["foo"] = x["foo"] || Hash.new # x["foo"] が空なら新規ハッシュ代入
  保存したい変数 = x["foo"]
    :
    :
end
```

PStoreクラスの利用法

「名前」と「ひとこと」を読み込んで出力するRubyプログラム：

```
#!/usr/koeki/bin/ruby
# coding: utf-8

word = Hash.new

print "名前は?: "
name = gets.chomp
print "ひとこと: "
word[name] = gets.chomp

for who, wd in word        # または word.each do |who, wd|
  printf("%sさんのひとこと「%s」\n", who, wd)
end
```

PStoreクラスの利用法

実行例：

```
% ./word.rb
名前は?: taro
ひとこと: hoge
taroさんのひとこと「hoge」
% ./word.rb
名前は?: hanako
ひとこと: やほー
hanakoさんのひとこと「やほー」
% ./word.rb
名前は?: John
ひとこと: Hey, Jude
Johnさんのひとこと「Hey, Jude」
```

最後でハッシュに含まれるキーと値のペアを全て出力しているが、プログラムを一度動かしても１つのキーと値しか登録されないので、１人の言葉しか出てこない。

PStoreクラスの利用法

PStoreを使って、そのときのハッシュ全体の値を別ファイル（以下の例ではword.db）に保存して毎回読み込むようにする。

```
#!/usr/koeki/bin/ruby
# coding: utf-8

require "pstore"
x = PStore.new("word.db")
x.transaction do
  x["word"] = x["word"] || Hash.new
# x["word"] ||= Hash.new でも可
  word = x["word"]

  print "名前は?: "
  name = gets.chomp
  print "ひとこと: "
  word[name] = gets.chomp

  for who, wd in word
    printf("%sさんのひとこと「%s」\n", who, wd)
  end
end
```

PStoreクラスの利用法

実行例：

```
% ./word.rb
名前は?: taro
ひとこと: hoge
tarooさんのひとこと「hoge」
% ./word.rb
名前は?: hanako
ひとこと: やほー
hanakoさんのひとこと「やほー」
tarooさんのひとこと「hoge」
% ./word.rb
名前は?: John
ひとこと: Hey, Jude
Johnさんのひとこと「Hey, Jude」
hanakoさんのひとこと「やほー」
tarooさんのひとこと「hoge」
```

以前に起動された値をword.dbファイルに自動的に保存し、前回の値を引き継ぐことができる。

CGIでの利用 (word-pstore-cgi.rb)

```
#!/usr/bin/env ruby
# coding: utf-8

myname="word-pstore-cgi.rb"
require "cgi"
c = CGI.new(:accept_charset => "UTF-8")
require "pstore"
x = PStore.new("data/word.db")       # 別ディレクトリにする

print "Content-type: text/html; charset=UTF-8\n\n"

print "<!DOCTYPE html>
<html>
<head><title>Word</title></head>
<body>"

# 値入力フォームもこのCGIで出力する。
# formのactionをこのCGIプログラムに指定している。
# (mynameはこのスクリプト名)
printf("<form method=\"POST\" action=\"./%s\">\n", myname)
print ' <p>
おなまえ: <input name="name" maxlength="40"><br>
ひとこと: <input name="word" maxlength="80"><br>
<input type="submit" value="GO">
<input type="reset" value="reset">
</p></form>'
```

```
x.transaction do
 x["word"] ||= Hash.new
 word = x["word"]
 if c["name"] > "" && c["word"] > ""
  name = c["name"]
  word[name] = c["word"]
 end
 print "<pre>\n"
 for p, w in word
  # フォーム入力値を出力するときは必ず CGI.escapeHTML() する
  person = CGI.escapeHTML(p)
  wrd = CGI.escapeHTML(w)
  printf("%sさんのひとこと「%s」\n", person, wrd)
 end
 print "</pre>"
end
puts "</body></html>"
```

データを保存するCGIの実行

CGIプログラムにデータファイルを書かせるためには当該ディレクトリを他人でも書き込めるように設定する。

1. データを書き込むディレクトリを別途作成する。

```
mkdir ~/public_html/mycgi/data
```

2. 誰にでも書き込みできる既存ファイルを消すのはファイル所有者のみ、という属性をディレクトリに設定する。

```
chmod 1777 ~/public_html/mycgi/data
```

3. ~/public_html/mycgi/にword-pstore-cgi.rbを保存し、chmod +xする。

実用CGIスクリプトへ

自分用データベース

PStoreクラスを利用して、何かの飲み物を飲んだときの感想を登録できるデータベースを作る。入力名として、

飲み物の名前	item
感想	comment

を利用する。

実用CGIスクリプトへ

入力フォームの出力と、現在登録されているデータの出力を両方とも行うようなCGIプログラムにする。

CGIプログラムの構成

1. HTMLヘッダ等の出力
2. データ入力用フォームの出力
3. その時点で何か値が入力されていたらそれをハッシュに追加
4. 既存データの値一覧を出力

cancoffee.rb

```
#!/usr/bin/env ruby
# coding: utf-8

require "cgi"
require "pstore"
myname = "cancoffee.rb"

c = CGI.new(:accept_charset => "UTF-8")
item = c["item"]
cmt  = c["comment"]
time  = Time.now # 時刻を保持するTimeクラス代入。nowは現時刻

print 'Content-type: text/html; charset=UTF-8

<!DOCTYPE html>
<html>
<head><title>飲んだものメモ</title>
<link rel="stylesheet" type="text/css" href="simple.css">
</head>
<body>
<h1>飲み物メモ</h1>
' # HTTPヘッダと冒頭部分
db = PStore.new("data/coffee.db")
db.transaction do # PStoreは db.transaction do ... end で使う
 db["root"] ||= Hash.new
 data = db["root"] # ここまではおきまり

 if item >"" && cmt > ""        # 名前とコメント、両方値があるなら登録
  data[item] = [time, cmt] # 今日の日付とコメント
 end

 # フォーム出力
  printf("<form method=\"POST\" action=\"./%s\">\n", myname)
  print '<p>
飲んだもの: <input name="item" type="text" maxlength="40"><br>
コメント <br>
<textarea name="comment" cols="40" rows="5">
</textarea><br>
<input type="submit" value="OK">
<input type="reset" value="reset"><br>
</p><hr>'

 # 既存のコメント出力(キー毎)
 print “<dl>\n” # 定義環境開始
 for i in data.keys.sort{|x, y|
    data[y][0] <=> data[x][0]  # 日付の新しい順にソート
  }
  day = data[i][0]  # 第0要素が日付
  msg = data[i][1] # 第1要素がコメント、それぞれ取り出す
  printf(" <dt>%s</dt>\n", i) # キー(つまり飲んだものの名前)
  printf(" <dd>記載日: %s<br>\n", day.strftime("%Y年%m月%d日"))
  printf("    %s</dd>\n", CGI.escapeHTML(msg))
 end
 print "</dl>\n” # 定義環境終了
end                  # db.transaction 終わり

print "</form><hr></body>\n</html>"
```

今回学ぶこと

1. 検索主体のCGIプログラムの作成
2. 紙芝居のようにページを進めるCGI
3. データを永続させるCGI
4. **CGIへの画像投稿**

CGIへの画像の投稿

HTMLフォームinput要素の**type = “file”**を利用することで、CGIへ画像を投稿できる。

multipart/form-data

文字列や選択オプションを送信するフォーム：

```
<form method="POST" action="./hoge.rb">
 一言メモ: <input name="var" size="40"><br>
 <input type="submit" value="SEND!">
 <input type="reset" value="reset">
</form>
```

multipart/form-data

ファイルの中身を投稿するには、フォームのenctype属性をmultipart/form-dataに変える。

```
<form method="POST" enctype="multipart/form-data"
action="./hoge.rb">
 一言メモ: <input name="var" size="40"><br><br>
 画像を選んでね: <input name="image" type="file">
 <input type="submit" value="SEND!">
 <input type="reset" value="reset">
</form>
```

multipart/form-data

HTMLでmultipart/form-dataを指定して入力させた値を受け取るCGIスクリプト：

```
require "cgi"
cgi = CGI.new(:accept_charset => "UTF-8")


memo = cgi["var"].read                          # CGI変数 var の受け取り(text)
photo_filename = cgi["image"].original_filename # ファイル名の受け取り
photo_size = cgi["image"].size                  # 送信ファイルサイズ取得
photo_data = cgi["image"].read                  # 送信ファイルデータの受け取り
```

実際のCGIフォームでの送信値を得るにはreadメソッドを介して行う。

multipart/form-data

読み取った画像データをファイルに保存：

```
open("ファイル名", "w") do |out|
  out.write cgi["image"].read
end
```

ファイルを保存するディレクトリはWebサーバ権限で書き込めるようにしておく(chmod 1777)。

今回学ぶこと

1. 検索主体のCGIプログラムの作成
2. 紙芝居のようにページを進めるCGI
3. データを永続させるCGI
4. CGIへの画像投稿
5. システム設計

自由課題説明

- 自由課題は好きな題材を考えてRubyプログラムを作る。
- チームメンバー全員でアイデアを出し、協力し合ってプログラムを作成する。
- 作ったプログラムの説明書も作成する。
- 春学期と同様roy上のWebページに記述する。

今後の予定

12月3日

- **チーム名と連絡係の決定**
- **作成システム案作成**

12月10日

- 仕様書の作成
- 骨格プログラムの完成と提出
- 作るプログラム（本命と押さえの２本）の決定
- 役割分担の決定

12月17日

- 予選発表

1月7日

- 代表発表

1月14日　全クラス合同発表会

係と仕事の分担

一人ひとりが以下の４つのうちから「主担当」となるもの、「副担当」となるものを１つ選ぶ。チーム内でなるべく人数の偏りがないようにする。

- プログラム作成
- データ作成（情報提示系）or ストーリー作成（体験提示系）
- 画面作成（HTML、画像、動画）
- 納品物作成（プレゼンシート＋レジュメ＋ポスター）
  納品物はいずれもPDF化してチームWebに置いてリンクする。

係と仕事の分担

係を決めるためにまず全員で要件定義を行う。
そのうち、作成するものについて４つの仕事に分けて進める。

1. 要件定義（全員で）
2. 画面作成
3. 機能設計
4. データ設計
5. 材料構築

テーマの設定

テーマは、

- 既存のサービスに**ないもの**
- 既存のサービスでは**行き届かない部分**が明らかにあるもの
- **受益者層**がはっきりしているもの
- 自分達が集める/作ることのが**できる**情報か（実行可能性/規模/権利関係）
- 具体的成果物を作る**動機**を高くもてるものか

要件定義

- 対象は「誰」の「どういう状況」
  - 実例を「Aさん」、「Bさん」の２パターン以上の物語で書いてみる。
  - 「具体的」にするのはその問題に関係する部分のみ（本筋無関係のところの具体性は脱線の元なので控える）。
  - 脱線しても否定はせず元に戻す働きかけをする。
- 何を提示する？「情報」or「体験」？
  - 「情報」の場合：実例を５個以上挙げてみる。
  - 「体験」の場合：動きを雑な絵に描いてみる。

以上が決まったら係ごとの作業に移る。

画面設計

- 何を入力させて
- 何を出力するか
- どこを見やすくするか
- 何枚の画面になるか
- デッサンする

データ設計者から５個のサンプルデータをもらって作成する。

機能設計

- 入力値をどう綺麗にするか（文字列か数値か）
- 出力の書式と文法の具体例
- 具体的にどう動かすか

データ設計者から５個のサンプルデータをもらって作成する。

データ設計

- 必須項目は何か
- 綺麗な形で格納する具体例
- どうやって手分けするか具体的手順

５個のサンプルデータをまず作り他の係に渡す。

材料

- 探す：オープンデータを探す
- 作る：自由に使えるツールを利用して作る（誰でも手軽に作れるようにする）

以上の作業が台無しにならないよう「著作権」に厳重注意して進める。

作業手順

1. テーマの選定（２つに絞る）**15分**
2. ストーリーの作成 **30分**
3. プロトタイプの作成
   - プログラム係：予備データで動くものを作る
   - 画面係：システムの入り口と操作する入力用Webページ
   - データ係：予備データを1人20個以上
   - 納品物作成：他の係の手伝い

本日は上記の2番まで行う。

授業内課題

チーム内で作成するシステム案を2つ選び、担当を割り当てる。

s4基礎プロII(F)の「#09 pf2水 授業内 (要件定義)」に指示どおり書き込む。提出後修正して良い。その場合は「編集」リンクから書き換えること。

今回学んだこと

1. 検索主体のCGIプログラムの作成
2. 紙芝居のようにページを進めるCGI
3. データを永続させるCGI
4. CGIへの画像投稿
5. システム設計

今回の目標

CGIを用いてWebインターフェースを設計できるようになる。

課題

A案、B案について「プログラム」「プロモーション Web」「データ（画像等を含む）」「スライド」「レジュメ」いずれかあるいは複数をメンバー間で均等に分担した「成果物バージョン0」を書き込む。

s4基礎プロII(F)の「#09 pf2水課題 (ver0)」に指示どおり書き込む。提出後修正して良い。その場合は「編集」リンクから書き換えること。

締切：**12月9日 (火)**

次回

第１回　生成AIを効果的に利用した学習法
第２回　メソッド定義と効果的活用（関数的処理）
第３回　メソッド定義と効果的活用（データ集合処理）
第４回　メソッド定義と効果的活用（再帰的アルゴリズム）
第５回　変数のスコープ・クラス設計
第６回　専門演習紹介
第７回　CGIと情報システム（1）
第８回　CGIと情報システム（2）
第９回　チーム課題準備期間（1）
第１０回　チーム課題準備期間（2）・trr試験
第１１回　予選発表
第１２回　代表発表
第１３回　合同成果発表会
第１４回　期末試験