result.py (資料編)

```
#!python
# -*- coding: shift_jis -*-

import sys , re                 #sysは関数の使用を可能にしたもの。reは正規表現の使用を可能にしたもの。

DOCTYPE = '<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">'
                                #'<!DOCTYPE HTML..."><'を変数DOCTYPEに代入する。

html = ""                       #空の変数htmlを用意する。
css = ""                        #空の変数cssを用意する。
css2 = ''                       #空の変数css2を用意する。
classno = 0                     #classnoを0に設定しておく。

######フォントサイズを変更するテーブル######
font_size_table = {
    "1":"x-small", "2":"small", "3":"medium", "4":"large", "5":"x-large", "6":"xx-large",
    "7":"150%", "-1":"small", "-2":"x-small", "-3":"xx-small", "+1":"large",
    "+2":"x-large", "+3":"xx-large"}

######HTMLの属性名をCSSの属性名を変更するテーブル######
attr_table = {
    "bgcolor":"background-color",
    "background":"background-image",
    "bgproperties":"background-attachment",
    "topmargin":"margin-top",
    "bottommargin":"margin-bottom",
    "leftmargin":"margin-left",
    "rightmargin":"margin-right",
    "text":"color",
    "align":"text-align",
    "type":"list-style-type",
    "font-face":"font-family",
    "font-color":"color",
    "table-align":"float",
    "size":"font-size",
    "font-weight":"font-style",
    "table align":"table{float:",
    "ul style":"ul {list-style-type",
    "ul type":"ul {list-style-type",
    "li style":"li {list-style-type",
    "li type":"li {list-style-type",
    "ol style":"ol {list-style-type",
    "ol type":"ol {list-style-type",
    "middle":"center",
    "bordercolor":"border-color"
    }



re_file_name = re.compile(u'¥.html?$')                                  #ファイル名を検索する。
re_tag_lower = re.compile(r'<.+?>')                                     #タグの中を検索する。
re_block_tag = re.compile(r'</(p|blockquote|ol|ul|dl|table|li|caption|dd|dt|head|td|tr|div|tbody)>')    #ブロックタグの終了タグを検索する。
re_block_tag2 = re.compile(r'<(hr|body|br).*?>')                        #ブロックタグを検索する。
re_spaces = re.compile(r' {2,}')                                        #半角スペースが二つ以上がある部分を検索する。
re_slash_tag = re.compile(u'</(b|i|u|s|strike|brink|strong)>')          #前項の終了タグを検索する。
re_endbody_tag = re.compile(u'</body>')                                 #</body>タグを検索する。
re_lattr = re.compile(u'<(h1|h2|h3|h4|h5|h6|tt|ul|li) ?(([A-Za-z]+) ?= ?"?(¥w+)"?)?>')                 #タグに属性名が一つのものを検索する。
re_body = re.compile(u'<body (.+?)>')                           #bodyタグを検索する。
re_many_attr = re.compile(u'<([0-9a-z]+) (.+?)>')                       #タグに属性名が一つのものを検索する。
re_properties = re.compile(u'([a-z]+) ?= ?(¥"([^¥"]+)¥"|([^ ]+))')      #属性名と属性の値を取り出す。
re_charset = re.compile(u'charset=(.+?)"', re.I)            #文字コードを検索する。。


######ファイルの文字を全て小文字に変換する######
def tag_lower(m):                               #tag_lowerという新しい関数を定義する。
    return m.group(0).lower()                       #戻り値　タグの中の全てを小文字に変換する。


######属性名が一つの場合######
def one_attr(m):                                #one_attrという新しい関数を定義する。
    global css, classno                             #global cssとclassnoの変数を用意する。
    tag = m.group(1)                                #タグ名を取り出し、変数tagに代入する。
    if m.group(2):                                  #もし、属性名があったら、
        if attr_table.has_key(m.group(3)):              #attr_tableの中にある属性名を取り出し、
            attr = attr_table[m.group(3)]                   #attrに属性名を代入する。
        else:                                       #attr_tableの中に属性名がなければ、
            attr = m.group(3)                           #attrにそのまま属性名を代入する。
        value = m.group(4)                          #valueに属性値を代入する。
        classno += 1                                #classnoを一つ足していく
        newclass = tag + "_" + value + str(classno)         #newclassにタグ (tag) と属性値(value) とナンバー(classno) を代入する。
        css += tag + '.' + newclass + "¥n{" + attr + ":" + value + ";} ¥n¥n"            #代入したものをcssに書き出す
        return '<' + tag + '  class="' + newclass + '">'    #戻り値　HTMLに<タグ名 class= "タグ名_属性名 ナンバー">を書き出す
```

```
    else:                                           #属性名がなかったら、
        return "<" + tag + ">"                          #戻り値　HTMLにそのまま<タグ名>を書き出す


######<body>に二つ以上の属性がある場合######
def body_each_property(m):                      #body_each_propertyという新しい関数を定義する。
    global css, css2                                #global cssとcss2の変数を用意する。
    if m.group(1):                                  #もし、属性名があったら、
        attr = m.group(1)                               #属性名を取り出し、変数attrに代入する。
        value = m.group(2)                              #属性値を取り出し、変数valueに代入する。
        if value[0] == '"':                             #もし、属性値に"があったら、
            value = value[1:-1]                             #属性値の"から"の中を取り出して、（２文字目から最後の手前まで）valueに代入する。
        if attr == 'background':                        #attrに'background'があれば、
            attr = 'background-image'                       #属性名を'background-image'に変更し、attrに代入する。
            value = 'url(' + value + ')';                   #urlと属性値をvalueに代入する。
        if attr == 'link':                              #もし、attrに'link'があれば、
            css2 += 'A:link { color : ' + value + ';}¥n'        #css2に属性名を変えて、書き出す。
        elif attr == 'alink':                           #もし、attrに'alink'があれば、
            css2 += 'A:active { color : ' + value + ';}¥n'      #css2に属性名を変えて、書き出す。
        elif attr == 'vlink':                           #もし、attrに'vlink'があれば、
            css2 += 'A:visited { color : ' + value + ';}¥n'     #css2に属性名を変えて、書き出す。
        else:                                           #attrが上記以外なら、
            if attr_table.has_key(attr):                    #attr_tableの中にある属性名のキーを取り出し、
                attr = attr_table[attr]                         #attrに属性名のキーを代入する。
            css += attr + ' : ' + value + ';¥n'             #cssに属性名と属性値を書き出す。
    return ''                                       #戻り値　変換せずにそのまま書き出す。


######<body>の変換######
def replace_body(m):                            #replace_bodyという新しい関数を定義する。
    global css, css2                                #global cssとcss2の変数を用意する。
    properties = m.group(1)                         #変数propertiesに属性名を代入する。
    css += 'body {¥n'                               #cssに'body {¥n'と書き出す。
    properties = re_properties.sub(body_each_property, properties)      #propertiesの属性名を関数body_each_property使って置換し、もう一度propertiesに
代入する。
    css += '}¥n'                                    #cssに'}¥n'と書き出す。
    if css2 != '':                                  #もし、css2に何か入っていたら
        css += css2 + '¥n'                              #cssにcss2と改行を代入して書き出す。
        css2 = ''                                       #css2は空にしておく。
    return '<body>'                                 #戻り値　'<body>'を書き出す。


######属性名が二つ以上がある場合######
def each_property(m):                           #each_propertyという新しい関数を定義する。
    global css                                      #global cssの変数を用意する。
    attr = m.group(1)                               #属性名を取り出し、attrに代入する。
    value = m.group(2)                              #属性値を取り出し、valueに代入する。
    if value[0] == '"':                             #もし、属性値に"があったら、
        value = value[1:-1]                             #属性値の"から"の中を取り出して、（２文字目から最後の手前まで）valueの変数に代入する。
        return ' ' + attr + '="' + value + '" '             #戻り値　半角スペースをあけて属性名と属性値を書き出す。
    if attr_table.has_key(attr):                    #attr_tableの中にある属性名のキーを取り出し、
        attr = attr_table[attr]                         #attrに属性名のキーを代入する。
    css += attr + ' : ' + value + ';¥n'             #cssに属性名と属性値を書き出す。
    return ''                                       #戻り値　変換せずにそのまま書き出す。


######テーブルタグに属性名がある場合######
def table_each_property(m):             #table_each_propertyという新しい関数を定義する。
    global css                              #global cssの変数を用意する。
    attr = m.group(1)                       #属性名を取り出し、attrに代入する。
    value = m.group(2)                      #属性値を取り出し、valueに代入する。
    if value[0] == '"':                     #もし、属性値に'"'があったら、
        value = value[1:-1]                     #属性値の"から"の中を取り出して、（２文字目から最後の手前まで）valueに代入する。
    if attr == 'border':                    #もし、attrに'border'があれば、
        value = value + 'px'                    #valueに属性名と単位を代入する。
        attr = 'border-width'                   #attrを'border-width'に変更する。
    elif attr == 'width':                   #もし、attrに'width'があれば、
        value = value + 'px'                    #valueに属性名と単位を代入する。
    elif attr == 'background':              #もし、attrに'background'があれば、
        attr = 'background-image'               #attrを'background-image'に変更する。
        value = 'url(' + value + ')';           #urlと属性値をvalueに代入する。
    elif attr == 'height':                  #もし、attrに'height'があれば、
        value = value + 'px'                    #valueに属性名と単位を代入する。
    elif attr == 'align' and value == 'middle':     #もし、attrが'align'で、valueが'middle'であれば、
        attr = 'vertical-align'                         #attrを'vertical-align'に変更する。
    elif attr_table.has_key(attr):          #attr_tableの中にある属性名のキーを取り出し、
        attr = attr_table[attr]                 #attrに属性名のキーを代入する。
    css += attr + ' : ' + value + ';¥n'     #cssに属性名と属性値を書き出す。
    return ''                               #戻り値　変換せずにそのまま書き出す。


#######フォントタグに属性名がある場合######
```

result.py（資料編）

```
def font_each_property(m):              #font_each_propertyという新しい関数を定義する。
    global css                              #global cssの変数を用意する。
    attr = m.group(1)                       #属性名を取り出し、attrに代入する。
    value = m.group(2)                      #属性値を取り出し、valueに代入する。
    if attr == 'style':                     #もし、属性名に'style'があれば、
        return ' ' + attr + '=' + value + ' '    #戻り値　半角スペースをあけて属性名と属性値を書き出す。
    if value[0] == '"':                     #もし、属性値に"があったら、
        value = value[1:-1]                     #属性値の"から"の中を取り出して、（２文字目から最後の手前まで）valueに代入する。
    if attr == 'size':                      #もし、属性名に'size'があれば、
        attr = 'font-size'                      #attrを'font-size'に変更する。
        if font_size_table.has_key(value):      #font_size_tableの中にある属性名のキーを取り出し、
            value = font_size_table[value]          #valueに属性値のキーを代入する。
        else:                                   #属性値のキーがなければ、
            value = "medium"                        #valueに"medium"を代入する。
    elif attr_table.has_key(attr):          #attr_tableの中にある属性名のキーを取り出し、
        attr = attr_table[attr]                 #attrに属性名のキーを代入する。
    css += attr + ' : ' + value + ';¥n'     #cssに属性名と属性値を書き出す。
    return ''                               #戻り値　変換せずにそのまま書き出す。


######属性名が二つ以上あるタグの変換######
def replace_many_attr(m):               #replace_many_attrという新しい関数を定義する。
    global css, classno                     #global cssとclassnoの変数を用意する。
    property = ''                           #空の変数"property"を用意する。
    tag = m.group(1)                        #タグ名を取り出し、tagに代入する。
    properties = m.group(2)                 #propertiesに属性名を代入する。
    if 'class=' in properties:              #もし、propertiesに'class='があれば、
        return '<' + tag + ' ' + properties + '>'       #戻り値　タグ名とpropertiesを書き出す。
    if tag == 'a' or tag == 'img' or tag == 'br' or tag == 'div' or tag == 'input' or tag == 'form' or tag == 'script':     #もしタグのなかに前項のタグがあったら
        return m.group(0)                               #戻り値　タグの中をすべて書き出す。
    if tag == 'table':                      #もし、propertiesに'table'があれば、
        css += '''                          #cssにtable, tr, th, td { border: 1px solid black;}と書き出す。
table, tr, th, td {
    border: 1px solid black;
}
'''
    classno +=1                             #クラスに1つナンバーを追加する。
    newclass = tag + "_" + str(classno)     #newclassにタグ名にclassnoを代入する。
    css += tag + '.' + newclass + ' {¥n'    #cssにタグ名とnewclass書き出す。
    if tag == 'font':                       #tagに'font'があれば、
        properties = properties.replace(u'ＭＳ ゴシック', 'serif')  #propertiesをユニコードに直し、 'serif'を'ＭＳ ゴシック'に変更する。
        properties = re_properties.sub(font_each_property, properties)  #propertiesにある属性名を関数font_each_propertyを使って置換し、propertiesに代入する。
    if tag == 'table' or tag == 'col' or tag == 'td' or tag == 'th':    #もし、タグに前項のタグがあれば、
        properties = re_properties.sub(table_each_property, properties) #propertiesにある属性名をtable_each_propertyの関数を使って置換し、もう一度propertiesに代入する。
    else:                                                       #それ以外は、
        properties = re_properties.sub(each_property, properties)   #propertiesにある属性名を関数each_propertyを使って置換し、propertiesに代入する
    css += '}¥n¥n'                          #cssを書き出す。
    return '<' + tag + ' class="' + newclass + '"' + property + '>' #戻り値　<タグ名 class = newclass + property>を書き出す。


############################################################
############################################################

input_file_name = sys.argv[1]                                   #変数input_file_nameにコマンドラインの引数で指定したファイル名（～.html=sys.argv[1]）を代入する。
output_file_name = re_file_name.sub(".x.html", input_file_name)     #input_file_nameに代入したファイル名を～.htmlのファイル名に置換し、変数output_file_nameに代入する。


n = 0                                   #変数nに0を代入しておく。
inhead = 0                              #変数inheadに0を代入しておく。
head = ''                               #空の変数"head"を用意する。
for line in open(input_file_name, 'rU'):    #input_file_nameをユニコードで読み込み、openし、一行ずつlineに代入する。
    line = line.rstrip()                    #line = line.rstrip()で、行末の改行を削除しておく。
    if line == '' and n == 0:               #もし、lineに何もなく、nに0が入っていたら、
        continue                                #実行をそこで打ち切って、 for の先頭から実行を続ける。
    n += 1                                  #nを一つ増やす。
    if '<body' in line or '<BODY' in line:  #もし、lineに'<body'か'<BODY'があれば、
        inhead = 0                              #変数のinheadに0を代入する。
    if n == 1:                              #もし、nが1だったら、
        inhead = 1                              #変数のinheadに1を代入する。
        if '<!DOCTYPE' in line:                 #もし、lineに'<!DOCTYPE'があれば、
            line = DOCTYPE                          #lineにDOCTYPEを代入する。
        else:                                   #それ以外は、
            line = DOCTYPE + '¥n' + line            #lineにDOCTYPEと改行、lineを代入する。
    elif inhead == 1:                       #もし、inheadが1だったら、
        line = re_tag_lower.sub(tag_lower, line)    #lineの中を全て小文字に置換し、lineに代入する。

    if inhead == 1 and ('content-style-type' in line or 'stylesheet' in line):
```

```
                                        #もし、inheadが1で、lineに'content-style-type'か'stylesheet'があれば、
        continue                        #実行をそこで打ち切って、 for の先頭から実行を続ける。
    if inhead == 1 and '</head>' in line:       #もし、inheadが1で、lineに</head>があれば、
        line = '''<meta http-equiv="Content-Style-Type" content="text/css">     #lineに<meta http-equiv="Content-Style-Type" content="text/css">
<link rel="stylesheet" type="text/css" href="style.css">                    #<link rel="stylesheet" type="text/css" href="style.css">と代入する。
''' + line
    if inhead == 1:                     #もし、inheadが1だったら、
        head += line + '¥n'                 #headにlineと改行を書き出す。
    else:                               #inheadが1以外なら、
        if line == '':                      #もし、lineに何も入っていなかったら、
            html += '¥n'                        #htmlに改行を書き出す。
        else:                               #それ以外は、
            html += line + ' '                  #htmlにlineと半角スペースを書き出す。

m = re_charset.search(head)             #headの中の文字コードが何であるか検索し、変数のmに代入する。
if m:                                   #もし、mがあったら、
    charset = m.group(1)                    #文字コードを変数のcharsetに代入する。
else:                                   #文字コードの指定がない場合、
    code = raw_input("文字コードを指定してください。番号を押して《Enter》を押してください。¥n"      #ユーザーに文字コードを指定してもらう。
"1.Shift_JIS ¥n2.euc-jp ¥n3.iso-2022-jp ¥n0.終了¥n")
    if code == "1":                     #もし、文字コードが'Shift_JIS'なら、
        charset = 'Shift_JIS'               #変数のcharsetに'Shift_JIS'を代入する。
    elif code == "2":                   #もし、文字コードが'euc-jp'なら、
        charset = 'euc-jp'                  #変数のcharsetに'euc-jp'を代入する。
    elif code == "3":                   #もし、文字コードが'iso-2022-jp'なら、
        charset = 'iso-2022-jp'             #変数のcharsetに'iso-2022-jp'を代入する。
    else:                               #それ以外なら、
        sys.exit(0)                         #Pythonを終了する。

head = unicode(head, charset)           #charsetに代入したHTMLファイルの文字コードをunicodeに変換し、headに書き出す。
html = unicode(html, charset)           #charsetに代入したHTMLファイルの文字コードをunicodeに変換し、htmlに書き出す。

html = re_tag_lower.sub(tag_lower, html)        #htmlの中を全て小文字に変換し、結果をhtmlに代入する。
html = re_body.sub(replace_body, html)          #htmlの中を関数replace_bodyで変換し、結果をhtmlに代入する。

if '<b>' in html:                               #もしhtmlファイルに<b>があったら、
    html = html.replace('<b>', '<span class="bold">')       #htmlの<b>を<span class="bold">に変更する。
    css += 'span.bold { font-weight: bold }¥n¥n'            #cssにspan.bold { font-weight: bold }を書き出す。
if '<i>' in html:                               #もしhtmlファイルに<i>があったら、
    html = html.replace('<i>', '<span class="italic">')     #htmlの<i>を<span class="italic">に変更する。
    css += 'span.italic { font-style: italic }¥n¥n'         #cssにspan.italic { font-style: italic }を書き出す。
if '<u>' in html:                               #もしhtmlファイルに<u>があったら、
    html = html.replace('<u>', '<span class="underline">')  #htmlの<u>を<span class="underline">に変更する。
    css += 'span.underline { text-decoration: underline }¥n¥n'  #cssにspan.underline { text-decoration: underline }を書き出す。
if '<s>' in html:                               #もしhtmlファイルに<s>があったら、
    html = html.replace('<s>', '<span class="strike">')     #htmlの<s>を<span class="strike">に変更する。
    css += 'span.strike { text-decoration: line-though }¥n¥n'   #cssにspan.strike { text-decoration: line-though }を書き出す。
if '<strike>' in html:                          #もしhtmlファイルに<strike>があったら、
    html = html.replace('<strike>', '<span class="strike-through">')        #htmlの<strikeb>を<span class="strike-through">に変更する。
    css += 'span.strike-through { text-decoration: line-through }¥n¥n'      #cssにspan.strike-through { text-decoration: line-through }を書き出す
if '<brink>' in html:                           #もしhtmlファイルに<brink>があったら、
    html = html.replace('<brink>', '<span class="brink">')      #htmlの<brink>を<span class="brink">に変更する。
    css += 'span.brink { text-decoration: brink }¥n¥n'          #cssにspan.brink { text-decoration: brink }を書き出す。
if '<strong>' in html:                          #もしhtmlファイルに<strong>があったら、
    html = html.replace('<strong>','<span class="bold">')       #htmlの<strong>を<span class="bold">に変更する。
    css += 'span.bold { font-weight: bold }¥n¥n'                #cssにspan.bold { font-weight: bold }を書き出す。
if '<center>' in html:                          #もしhtmlファイルに<center>があったら、
    html = html.replace('<center>','<div align="center">')      #htmlの<center>を<div align="center">に変更する。
    css += 'div{text align: center }¥n¥n'                       #cssにdiv{text align: center }を書き出す。
    html = html.replace('</center>', '</div>')                  #htmlの</center>を</div>に変更する。

html = re_slash_tag.sub('</span>', html)        #htmlの中の終了タグを'</span>'に置換し、htmlに代入する。
html = re_endbody_tag.sub('</style>', html)     #htmlの中の</body>タグを'</style>'に置換し、htmlに代入する。
html = re_block_tag.sub('¥g<0>¥n', html)        #htmlの中の属性がないブロックタグに改行を挿入し置換して、htmlに代入する。
html = re_block_tag2.sub('¥g<0>¥n', html)       #htmlの中の属性があるブロックタグに改行を挿入し置換して、htmlに代入する。
html = re_spaces.sub(' ', html)                 #htmlの中の二つ以上空白がある部分に半角スペースを入れて置換し、htmlに代入する。
html = re_1attr.sub(one_attr, html)             #htmlの中の属性名が一つのものがあれば、関数one_attrに入れて置換し、htmlに代入する。


html = re_many_attr.sub(replace_many_attr, html)    #htmlの中の属性名が二つ以上のものがあれば、関数replace_many_attrに入れて置換しhtmlに代入する。



css_file = open('style.css', 'w')               #style.cssを書き込みモードでopenし、css_fileに代入する。
css = css.encode(charset)                       #cssの文字コードをunicodeから変換前のファイルの文字コードになおして、cssに代入する。
css_file.write(css)                             #cssをcss_fileに書き出す。
css_file.close()                                #style.cssを閉じる。


head = head.encode(charset)                     #headの文字コードをunicodeから変換前のファイルの文字コードになおして、headに代入する。
html = html.encode(charset)                     #htmlの文字コードをunicodeから変換前のファイルの文字コードになおして、htmlに代入する。
output_file_name = open('style.html','w')       #style.htmlを書き込みモードでopenし、output_file_nameに代入する。
```

```
output_file_name.write(head + html)                #headとhtmlをoutput_file_nameに書き出す。
output_file_name.close()                           #output_file_nameを閉じる。

sys.exit()                                         #pythonを終了させる
```

result.py（資料編）

『古いHTMLの再利用について－CSS化ツールの制作－』

このツールはデザイン要素（<b>や<body bgcolor=red>など）を含んでいるHTMLファイルの中を文書の構成はHTMLファイル、デザイン要素はCSSファイルと別々のファイルとして書き出されるように変換するツールである。

《使い方》
１．WindowsにPythonが入っていない場合、"http://tibet.que.ne.jp/otani/2005program/install.html"からPythonをインストールしておく。（knoppixを使用する場合はインストールしなくてよい。）

２．デザイン要素（<b>や<body bgcolor=red>など）を含んでいるHTMLファイルを一つ用意し、ファイル名をtest.htmlとして保存しておく。
（見つけられない場合には資料のCD-Rに"test.html"がある）

《注意》
このツールを使用する前に、"http://htmllint.itc.keio.ac.jp/htmllint/htmllintl.html"でチェックしたいHTMLのURLを指定するか、HTMLを下のテキスト領域に直接記述して、［チェック］ボタンを押す。［リセット］は全ての設定内容を初期状態に戻す。［クリア］はそれぞれの内容を消去する。このHTML文書の文法をチェックで90点以上とったファイルのみを変換してほしい。

３．資料 "result.py"を(z:)ドライブにコピーする。

４．コマンドプロンプトを起動させる。
（スタート→アクセサリ→コマンドプロンプト）

５．（大谷大学内でPythonを使用する場合、コマンドプロンプトでsetenvと打ってEnterを押す。）
それ以外は、python result.py test.pyと打ってEnterを押し、Pythonを実行させる。

６．test.pyを保存した同じフォルダにstyle.htmlとstyle.cssの2つのファイルが表示されている。
style.htmlは変換前のtest.htmlの文書の構成のみを書き出したものである。
style.cssは変換前のtest.htmlの中にあったデザイン要素のみを書き出したものである。

７．style.htmlについては、Internet ExplorerまたはfireFoxで変換前のHTMLファイルと画面表示が同じように変換されているか確認する。
またstyle.htmlとstyle.cssともに、エディタ（サクラエディタ、秀丸、emacsなど）で開き、style.htmlでタグがきちんと変換されているか確認する。style.cssについては、タグや属性名、属性値にデザインを追加し、反映されるか確認する。