情報工学実験1

「スクリプトプログラミング」

担当教員：　　當間 愛晃
学籍番号：　０９５７０７Ｂ
氏　　名：　大 城 佳 明
提 出 日：　2010年5月18日

目 次

1　Level 1：四則演算スクリプトの作成

1.1　課題説明

1. a=8, b=2 とし、四則演算を計算する。

1.2　sh 作成

1.2.1　スクリプト本体 (Level1.sh)

作成したスクリプト Level1.sh を以下に示す

```
01　#!/bin/sh
02　echo ”# Level1 の実行結果”
03　a=8
04　b=2
05　echo ”a=$a, b=$b”
06　answer=‘expr $a + $b‘
07　echo ”a+b=$answer”
08　answer=‘expr $a - $b‘
09　echo ”a-b=$answer”
10　answer=‘expr $a \* $b‘
11　echo ”a*b=$answer”
12　answer=‘expr $a / $b‘
13　echo ”a/b=$answer”
```

1.2.2　実行結果 (Level1.sh)

level1.sh を実行した結果を示す.

```
01　# Level1 の実行結果
02　a=8, b=2
03　a+b=10
04　a-b=6
05　a*b=16
06　a/b=4
```

1.2.3　考察 (Level1.sh)

1. 文字出力は「echo ”〜”」である。
2. 「$引数」で引数を使うことが出来る。
3. 計算式は `` で囲まなければならない
4. あと計算には「expr」が必要である
5. 「+，-，/」を使うことにより、「加算、減算、除算」が出来る。
6. 「*」は特殊文字なので乗算する時は「 *」でなければならない。

1.3 Python 作成

1.3.1 スクリプト本体 (Level1.py)

作成したスクリプト Level1.py を以下に示す

```
01  # coding: UTF-8
02  print "# Level1 の実行結果"
03  a=8
04  b=2
05  print "a=" + str(a) + "b=" + str(b)
06  print "a+b=",a+b
07  print "a-b=",a-b
08  print "a*b=",a*b
09  print "a/b=",a/b
```

1.3.2 実行結果 (Level1.py)

level1.py を実行した結果を示す

```
01  # Level1 の実行結果
02  a=8, b=2
03  a+b=10
04  a-b=6
05  a*b=16
06  a/b=4
```

1.3.3 考察 (Level1.py)

1. 01 は使う文字の種類である
2. 文字出力は「print "〜"」である
3. 「str(引数)」で引数を出力することが出来る。
4. " 〜 " で囲まなければ四則演算が出来る
5. 実行結果より sh の時と同じように出来たことがわかる

2 Level 2：引数を受け取る四則演算スクリプトの作成

2.1 課題説明 (Level2.sh)

1. level1.sh を拡張（複製して新しくファイルを作成）する。
2. level1.sh では演算対象の a,b をスクリプト内で設定していたが、この 2 つの値を実行時の引数から設定するようにする。

2.2 sh 作成

2.2.1 スクリプト本体 (Level2.sh)

作成したスクリプト Level2.sh を以下に示す

```
01  #!/bin/sh
02  echo "# Level2 の実行結果"
03  a=$1
04  b=$2
05  echo "a=$a, b=$b"
06  answer=`expr $a + $b`
07  echo "a+b=$answer"
08  answer=`expr $a - $b`
09  echo "a-b=$answer"
10  answer=`expr $a \* $b`
11  echo "a*b=$answer"
12  answer=`expr $a / $b`
13  echo "a/b=$answer"
```

2.2.2 実行結果 (Level2.sh)

level2.sh を実行した結果を示す.

```
01  yoshiaki-oshiro-no-macbook-2:jikken yoshiaki$ ./Level2.sh 8 2
02  # Level2 の実行結果
03  a=8, b=2
04  a+b=10
05  a-b=6
06  a*b=16
08  a/b=4
```

2.2.3 考察 (Level2.sh)

1. 実行時に入力した数字の 1 つ目が「$1」、２つ目が「$2」である
2. 実行結果 01,03 より実行時に入力した値「$1 = 8」「$2 = 2」が使われているのがわかる。
3. 実行結果より answer の値は上書きされていることがわかる
4. 他は Level1.sh と同様である

2.3 Python 作成

2.3.1 スクリプト本体 (Level2.py)

作成したスクリプト Level2.py を以下に示す

```
01  # coding: UTF-8
02  import sys
03  a=sys.argv[1]
04  b=sys.argv[2]
05  print "# Level2 の実行結果"
06  print "a=" + str(a) + " b=" + str(b)
07  print "a+b=",a+b
08  print "a-b=",a-b
09  print "a*b=",a*b
10  print "a/b=",a/b
```

2.3.2 実行結果 (Level2.py)

level2.py を実行した結果を示す

```
01  yoshiaki-oshiro-no-macbook-2:jikken yoshiaki$ python Level2.py 8 2
02  # Level2 の実行結果
03  a=8, b=2
04  a+b= 82
05  a-b=
06  Traceback (most recent call last):
07  File "Level2.py", line 8, in <module>
08  print "a-b=",a-b
```

2.3.3 考察 (Level2.py)

1. コマンドラインで入力するためには「import 引数」が必要である。
2. 引数に入れた値は配列となっている
3. 配列 [1] が 8、配列 [2] が 2 である
4. 実行結果まで 03 までは成功したが 04 からおかしいことになっている
5. 実行結果 04 より答えが「10」のはずなのに、「82」になっている
6. これは文字「8」と文字「2」の足し算で 04 のような結果になったと思われる
7. 文字「8」と文字「2」の「 - , * , / 」は出来ないのでエラーが起きた

2.3.4 スクリプト本体 (Level2-1.py)

作成したスクリプト Level2-1.py を以下に示す

```
01  # coding: UTF-8
02  import sys
03  a=int(sys.argv[1])
04  b=int(sys.argv[2])
05  print "# Level2-1 の実行結果"
06  print "a=" + str(a) + " b=" + str(b)
07  print "a+b=",a+b
08  print "a-b=",a-b
09  print "a*b=",a*b
10  print "a/b=",a/b
```

2.3.5 実行結果 (Level2-1.py)

level2-1.py を実行した結果を示す

```
01  yoshiaki-oshiro-no-macbook-2:jikken yoshiaki$ python Level2-1.py 8 2
02  # Level2-1 の実行結果
03  a=8 b=2
04  a+b= 10
05  a-b= 6
06  a*b= 16
07  a/b= 4
```

2.3.6 考察 (Level2-1.py)

1. スクリプト本体 03,04 のところを書き換えた
2. 「int(配列)」で文字を数字に変換出来た。
3. 文字を数字に変化することで計算出来るようになる
4. 実行結果より、すべて計算出来たことがわかる

3　Level 3：int 型の引数を 2 個取り、数値の比較

3.1　課題説明 (Level3.sh)

1. int 型の引数を 2 個取り、それぞれ int1, int2 として設定 (保存/代入) する。
2. 引数が 2 個以外の場合（0 個、1 個や、3 個以上）には使い方を出力して終了する。
3. int1 と int2 を数値比較し、以下のように出力すること。

3.2　sh 作成

3.2.1　スクリプト本体 (Level3.sh)

作成したスクリプト Level3.sh を以下に示す

```
01  #!/bin/sh
02  #————入力したのが２つかどうかの判断————-#
03  kazu=$#
04  if [ $kazu -eq 2 ] ; then
05      int1=$1
06      int2=$2
07  else
08      echo ”./level3.sh int1 int2”
09      exit 1
10  fi
11  #————数値比較————#
12  echo ”int1=$int1, int2=$int2”
13  #————値が同じのとき————#
14  if [ $int1 -eq $int2 ] ; then
15      echo ”int1 is equql to int2”
16  fi
17  #————int1 の値が大きいとき————#
18  if [ $int1 -gt $int2 ] ; then
19      echo ”int1 is greater than int2”
20  fi
21  #————int1 の値が小さいとき————#
22  if [ $int1 -lt $int2 ] ; then
23      echo ”int1 is less than int2”
24  fi
```

3.2.2 実行結果 (Level3.sh)

level3.sh を実行した結果を示す.

```
01  yoshiaki-oshiro-no-macbook-2:jikken yoshiaki$ ./Level3.sh
02  ./level3.sh int1 int2

03  yoshiaki-oshiro-no-macbook-2:jikken yoshiaki$ ./level3.sh 10 20
04  int1=10, int2=20
05  int1 is less than int2

06  yoshiaki-oshiro-no-macbook-2:jikken yoshiaki$ ./level3.sh 10 10
07  int1=10, int2=10
08  int1 is equql to int2

09  yoshiaki-oshiro-no-macbook-2:jikken yoshiaki$ ./level3.sh 20 10
10  int1=20, int2=10
11  int1 is greater than int2
```

3.2.3 考察 (Level3.sh)

1. 数値の比較は if 文で行う
   (a) 03〜09 は入力した値が２つかどうかの判定するプログラムである
   (b) 05,06 より入力した値が２つならそれぞれに引数を与えている
   (c) 08 より ２つではなかった「./level3.sh int1 int2」が表示される
   (d) 09 はプログラム自体の強制終了の意味である
2. 14〜24 が数値の比較を行う
   (a) 「-eq」は「＝」と同じ意味
   (b) 15 では２つの値が同じなら「int1 is equql to int2」と出力する
   (c) 「-gt」は「>」と同じ意味
   (d) 19 では int1 の値の方が大きいなら「int1 is greater than int2」と出力する
   (e) 「-le」は「<」と同じ意味
   (f) 23 では int1 の値の方が小さいなら「int1 is less than int2」と出力する
3. 実行結果より正しく出力されていることがわかる

3.3 Python 作成

3.3.1 スクリプト本体 (Level3.py)

作成したスクリプト Level3.py を以下に示す

```
01  #conding:UTF-8
02  import sys
03  a=sys.argv[1]
04  b=sys.argv[2]
05  if a == None or b == None :
06      print "python level3.py int1 int2"
07  else:
08      print "a=" +str(a) + " b=" +str(b)
09      if a == b :
11          print "int1 is equql to int2"
12      if a > b :
13          print "int1 is greater than int2"
13      if a < b :
14          print "int1 is less than int2"
```

3.3.2 実行結果 (Level3.py)

level3.py を実行した結果を示す

```
01  yoshiaki-oshiro-no-macbook-2:jikken yoshiaki$ python Level3.py
02  Traceback (most recent call last):
03  File "Level3.py", line 3, in <module>
04  a=sys.argv[1]
05  IndexError: list index out of range

06  yoshiaki-oshiro-no-macbook-2:jikken yoshiaki$ python Level3.py 10 20
07  a=10 b=20
09  int1 is less than int2

10  yoshiaki-oshiro-no-macbook-2:jikken yoshiaki$ python Level3.py 10 10
11  a=10 b=10
12  int1 is equql to int2

13  yoshiaki-oshiro-no-macbook-2:jikken yoshiaki$ python Level3.py 20 10
14  a=20 b=10
15  int1 is greater than int2
```

3.3.3 考察 (Level3.py)

1. スクリプト本体 05 より a、b の値 (2つ入力) があるか判断している
2. a または b の値が存在しないなら「python level3.py int1 int2」と出力される
3. しかし a、b の値がない場合にエラーが起きている

4. エラー内容から見ると、配列が存在しないものを代入してるのでエラーが起きている
5. この方法だと出来ないことがわかる
6. ２つある場合は問題なく出力されている
7. python ではスペースを一つ範囲が if 文が実行する範囲である
8. 論理式は C 言語と変わらない

3.3.4 スクリプト本体 (Level3-1.py)

作成したスクリプト Level3-1.py を以下に示す

```
01  #conding:UTF-8
02  import sys
03  kazu = len(sys.argv)
04  if kazu == 3 :
05      a=int(sys.argv[1])
06      b=int(sys.argv[2])
07      print "a=" +str(a) + " b=" +str(b)
08      if a == b :
09          print "int1 is equql to int2"
10      if a > b :
11          print "int1 is greater than int2"
12      if a < b :
13          print "int1 is less than int2"
14  else:
15      print "python level3.py int1 int2"
```

3.3.5 実行結果 (Level3-1.py)

level3-1.py を実行した結果を示す

```
01  yoshiaki-oshiro-no-macbook-2:jikken yoshiaki$ python Level3-1.py
02  python level3.py int1 int2

03  yoshiaki-oshiro-no-macbook-2:jikken yoshiaki$ python Level3-1.py 10 20
04  a=10 b=20
05  int1 is less than int2

06  yoshiaki-oshiro-no-macbook-2:jikken yoshiaki$ python Level3-1.py 10 10
07  a=10 b=10
08  int1 is equql to int2

09  yoshiaki-oshiro-no-macbook-2:jikken yoshiaki$ python Level3-1.py 20 10
10  a=20 b=10
11  int1 is greater than int2
```

3.3.6　考察 (Level3.py)

1. スクリプト本体 03 より配列の中の要素数を求めている
2. len(〜) で要素数を数えることができる
3. 「Level3-1.py」「int1」「int2」より 3 つの要素数になる
4. 03 より要素数が 3 ならば数値の比較を行う
5. 実行結果より成功していることがわかる

4 Level 4：ディレクトリの表示

4.1 課題説明

1. 指定したディレクトリの内容を表示するコマンド myls.sh を作成せよ。
2. 指定されたディレクトリが存在しない場合には「Not found: $dir」等のようにエラーを出力して終了する事。
3. 指定されたディレクトリ内に存在するディレクトリと通常のファイルは、出力を分けて表示すること。
4. ディレクトリの後ろには ”/” を表示すること。

4.2 スクリプト本体 (myls.sh)

作成したスクリプト myls.sh を以下に示す

```
01  #!/bin/sh
02  #―――――-スクリプトに渡された引数の個数を求める―――――-#
03  kazu=$#
04  #―――――-引数が 1 つかどうか調べる―――――-#
05  if [ $kazu -eq 1 ] ; then
06      dir=$1
07  else
08      echo ”引数が 1 つ必要で、ディレクトリを指定してください”
09      exit 1
10  fi
11  #―――――-指定した場所がディレクトリかどうかの判断―――――-#
12  if [ -d $dir ] ; then
13  #―――――-ファイルの取得―――――-#
14      filelist=‘/bin/ls -1 $dir‘
15  #―――――-ディレクトリかどうか調べて表示する―――――-#
16      for file in $filelist
17      do
18          if [ -d $dir/$file ] ; then
19              echo ”$file/”
20          fi
21  #―――――-ファイルかどうか調べて表示する―――――-#
22      for file in $filelist
23      do
24          if [ -f $dir/$file ] ; then
25              echo ”$file”
26          fi
27      done
28  #―――――-指定した場所はディレクトリではない場合―――――-#
29  else
30      echo ”Not found: $dir”
31      exit 1
32  fi
```

4.3 実行結果 (myls.sh)

level4.sh を実行した結果を示す.

```
01  yoshiaki-oshiro-no-macbook-2:jikken yoshiaki$ ./Level4.sh .
02  af/
03  afaw/
04  b/
05  c/
06  fa/
07  jikkne/
08  #Level2-1.py#
09  #hello.py#
10  Level1.py
11  Level1.py
12  Level1.sh
13  Level1.sh
14  Level2-1.py
15  Level2-1.py
17  Level2.py
                    < 省略 >
```

4.4 考察 (myls.sh)

4.4.1 説明

1. 最初に渡された引数の数が 1 つかどうかを調べている
2. スクリプト本体 12 の 「 -d $dir」で渡された場所はディレクトリかどうかを判断している
3. スクリプト本体 14 の「‘ -1 $引数 ‘」より指定した場所のデータをすべて入れる
4. スクリプト本体 16,22 の「for file in $fileslit」はファイルがある限り実行されるという意味である
5. 「$file/」よりディレクトリには「/」を後ろにつけていることがわかる
6. 実行結果 01〜07 より「/」がついているのでディレクトリであることがわかる

4.4.2 利点・欠点

1. 利点
   (a) ディレクトリとファイルを分けることで探しやすい
   (b) 「/」をつけることでディレクトリかどうが判断しやすい
2. 欠点
   (a) 数が多くなると縦に長くなり見づらい
   (b) 数が多くなるとディレクトリとファイルの分岐点がわかりづらい

4.5 スクリプト本体 (myls-1.sh)

1. さらに見た目を考慮したプログラムの作成した
2. 作成したスクリプト myls-1.sh を以下に示す。
3. 変更した部分だけ示す

```
13  #―――-ファイルの取得―――-#
14  filelist=‘/bin/ls -1 $dir‘
15  cdir=0
16  cfil=0
17  #―――-ディレクトリかどうか調べて表示する―――-#
18  echo ”-directory――――――――――――――――――――――――――――――”
19  for file in $filelist
20  do
21      if [ -d $dir/$file ] ; then
22          echo "\e[33m ${file}/ \e[m \c"
23          cdir=‘expr $cdir + 1‘
24          kai=‘expr $cdir % 5‘
25  #―――- 1 行のディレクトリの数が 5 個目なら改行する―――-#
26          if [ $kai -eq 0 ] ; then
27              echo ” ”
28              fi
29          fi
30      done
31      echo ” ”
32      echo ”-file――――――――――――――――――――――――――――――――”
33  #―――-ファイルかどうか調べて表示する―――-#
31      for file in $filelist
32      do
33      if [ -f $dir/$file ] ; then
34          echo "\e[34m ${file} \e[m \c"
35          cfil=‘expr $cfil + 1‘
36          kai=‘expr $cfil % 5‘
37  #―――- 1 行のファイルの数が 5 個目ならば改行する―――-#
38          if [ $kai -eq 0 ] ; then
39              echo ” ”
40              fi
41          fi
42  done
43  echo ” ”
44  #―――-合計を表示―――-#
45  echo ”-toal――――――――――――――――――――――――――――――――”
46  echo "\e[31m directory：${cdir}          file：${cfil} \e[m"
47  #―――-指定した場所はディレクトリではない場合―――-#
```

4.6 実行結果 (myls-1.sh)

level4.sh を実行した結果を示す.

```
01  -directory----------------------------------------
02  af/  afaw/  b/  c/ fa/
03  jikkne/
04  -file---------------------------------------------
05  #Level2-1.py#  #hello.py#  Level1.py  Level1.py  Level1.sh
06  Level1.sh  Level2-1.py  Level2-1.py  Level2.py  Level2.py
07  Level2.sh  Level2.sh  Level3-1.py  Level3-1.py  Level3.py
08  Level3.py  Level3.sh  Level3.sh  Level4-1.sh  Level4-1.sh
09  Level4.py  Level4.py  fara.java  hello.py  hello.py
10  level4.sh  level4.sh  myls-1.sh  myls-1.sh  myls.sh
11  myls.sh  na.c  test.sh  test.sh
12  -toal---------------------------------------------
13  directory：6  file：34
```

4.7 考察 (myls-1.sh)

4.7.1 スクリプト本体

1. 15,16 にカウンター用に「cdir」「cfil」を用意した
2. 22,34 の説明をする
   (a) 「\e[数字 m 〜 \e[m」で色の設定をしている
   (b) 33 は黄色、34 は青、31 は赤である
   (c) ”〜 \c”で改行をなくしている
3. 23,35 はカウンタである
4. 26〜28,38〜40 は改行をするためのプログラミングである

4.7.2 実行結果

1. 1 行 5 個で表示されてる
2. 02,03 はディレクトリなので黄色で表示されている
3. 05〜11 はファイルなので青で表示されている
4. 13 はディレクトリとファイルの合計が表示されてる

5 Level 7：gnuplot をスクリプトファイルを通して利用せよ

5.1 課題説明

1. 条件 0： gnuplot へのコマンド受け渡しをスクリプトファイルとする事。
2. 条件 1： 数値データ数を 10 件以上とする事
3. 条件 2： LaTeX の出力に含めるために、gnuplot の出力画像形式を EPS 形式とする事
4. 条件 3： 軸の説明やグラフのタイトルを図内に含める事

5.2 スクリプト本体 (command.plot.sh)

作成したスクリプト command.plot.sh を以下に示す

```
01  #!/bin/sh
02  #————出力画像の形式を設定．ここでは eps で出力する————#
03  echo 'set terminal postscript eps color' > Level7.gnuplot
04  #————出力ファイル名を決定————#
05  echo 'set output "Level7.eps"' >> Level7.gnuplot
06  #————グラフのタイトルを設定————#
07  echo 'set title "Mean air temperature of Okinawa(classified by month)"' >> Level7.gnuplot
08  #————x 軸のラベルを設定————#
09  echo 'set xlabel "Month (2009)"' >> Level7.gnuplot
10  #————y 軸のラベルを設定———-#
11  echo 'set ylabel "Air temperature"' >> Level7.gnuplot
12  #————y 軸に 2 カラム目（月），x 軸に 1 カラム目（気温）をプロット———-#
13  echo 'plot "Level7.dat" using 2:1 with linespoints' >> Level7.gnuplot
14  gnuplot < Level7.gnuplot
15  #——end for gnuplot
```

5.3 データファイル (Level7.dat)

作成したスクリプト Level7.dat を以下に示す

```
01  17.5 1/DC/2009
02  18.0 2/DC/2009
03  19.0 3/DC/2009
04  22.5 4/DC/2009
05  25.0 5/DC/2009
06  27.0 6/DC/2009
07  29.5 7/DC/2009
08  29.4 8/DC/2009
09  28.0 9/DC/2009
10  25.5 10/DC/2009
11  23.0 11/DC/2009
12  19.5 12/DC/2009
```

5.4 実行結果 (command.plot.sh , Level7.dat)

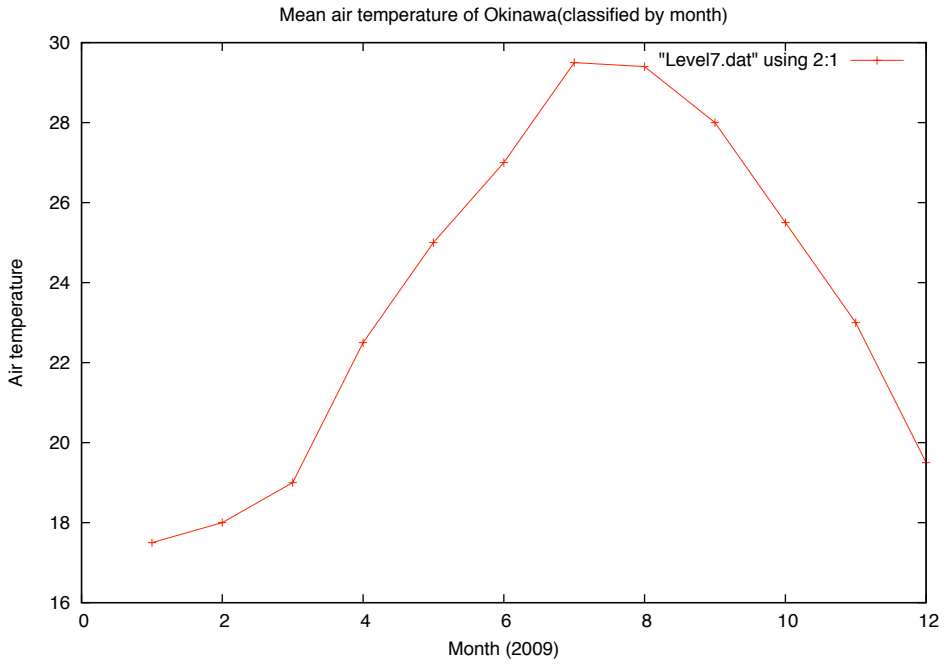
command.plot.sh を実行した結果を示す.

```
01  yoshiaki-oshiro-no-macbook-2:c yoshiaki$ ls
02  Level7.dat command.plot.sh

03  yoshiaki-oshiro-no-macbook-2:c yoshiaki$ ./command.plot.sh

04  yoshiaki-oshiro-no-macbook-2:c yoshiaki$ ls
05  Level7.dat Level7.eps Level7.gnuplot command.plot.sh

06  yoshiaki-oshiro-no-macbook-2:c yoshiaki$ open Level7.eps
```



5.5 考察 (command.plot.sh , Level7.dat)

1. 沖縄の年間平均気温についてのグラフを作成した
2. スクリプト本体 03,06 は eps で出力するためのプログラムである
3. スクリプト本体 18 の説明をする
   (a) Leve.dat を使用する
   (b) 「2:1」は 2 列目を X 軸, 1 列目を Y 軸をしている
   (c) グラフの形式は「linespoints」である
   (d) データファイルは 2 列の文字列で出来ている
   (e) 「 / 」より右側はメモである

4. 実行結果 01 より最初は２つファイル存在している

5. 実行結果 03 よりシェルを実行

6. 実行結果 04,05 より新しく「Level7.eps」「Level7.gnuplot」が作成出来たのがわかる

7. 実行結果 06 より図が正しく出力されているのがわかる

6　Level X：コピーファイルの作成

6.1　オプション課題説明

1. 指定したファイルのコピーを行う
2. ファイル名は「ファイル名-(数字) . 拡張子」で保存される

6.2　スクリプト本体 (LevelX-1.sh)

作成したスクリプト LevelX-1.sh を以下に示す

```
01  #!/bin/sh
02  #————————スクリプトに渡された引数の数を数える————————-#
03  kazu=$#
04  if [ $kazu -eq 2 ] ; then
05      ext1=$1
06      ext2=$2
07  else
08      echo ”引数 2 個必要（ファイル名, 拡張子）”
09      exit 1
10  fi
11  #————————現在の場所のファイル名を取得————————-#
12  filelist=‘/bin/ls‘
13  #————————ファイルが存在するかの判断————————-#
14  for file in $filelist
15      do
16      if [ $file = $ext1 ] ; then
17        #————————拡張子を取り除いた文字列を求める————-#
18        name=‘basename $ext1 .$ext2‘
19        num=1
20        #————————ファイル名に重複がないか判断————————#
21        for file in $filelist
22        do
23          if [ $file = $name-$num.$ext2 ] ; then
24            num=‘expr $num + 1‘
25          fi
26        done
27        #————————ファイルをコピー————————#
28        cp $ext1 $name-$num.$ext2
29        echo ”ファイル名 [ $name-$num.$ext2 ] でコピーしました”
30        exit 1
31      fi
32  done
33  echo ”ファイルが存在しません”
```

6.3 実行結果 (LevelX-1.sh)

LevelX-1.sh を実行した結果を示す.

```
01  yoshiaki-oshiro-no-macbook-2:b yoshiaki$ ls
02  LevelX-1-1.sh LevelX-1.sh LevelX-1.sh  text.sh

03  yoshiaki-oshiro-no-macbook-2:b yoshiaki$ ./LevelX-1.sh text.sh sh
04  ファイル名 [ text-1.sh ] でコピーしました

05  yoshiaki-oshiro-no-macbook-2:b yoshiaki$ ./LevelX-1.sh text.sh sh
06  ファイル名 [ text-2.sh ] でコピーしました

07  yoshiaki-oshiro-no-macbook-2:b yoshiaki$ ./LevelX-1.sh text.sh sh
08  ファイル名 [ text-3.sh ] でコピーしました

09  yoshiaki-oshiro-no-macbook-2:b yoshiaki$ ./LevelX-1.sh Level1.sh sh
10  ファイルが存在しません

11  yoshiaki-oshiro-no-macbook-2:b yoshiaki$ ls
12  LevelX-1-1.sh LevelX-1.sh  text-2.sh text.sh
13  LevelX-1.sh text-1.sh text-3.sh
```

6.4 考察 (LevelX-1.sh)

6.4.1 説明

1. 流れ（./LevelX-1.sh text.sh sh の時）
   (a) 入力された値が２つかどうかの判断（２つ表示されているので 04〜06）
   (b) 全てのファイル名を取得し、同じファイルが存在するか判断（14〜32 の for 文）
   (c) 新しいファイル名を作成し、重複がないかの判断を行う
   (d) 重複がある場合は新しいファイル名を作成して、もう一度判断させる
   (e) それを延々と続けることにより、いずれファイル名が重複しないファイル名が作成される
2. スクリプト本体
   (a) 02〜10 では Level4.sh とほぼ同じである
   (b) 12 で初めにファイル名を取得している
   (c) 14〜32 は指定したファイル名が存在するかを調べている
   (d) 16〜31 は存在した場合のパターンである。それについて詳しく説明する
      i. 18 では拡張子を取り除いたファイル名を取得している
      ii. 「basename 引数 1 引数 2」で引数 1 の文字列から引数 2 の文字列を引いている
      iii. 19num はファイルが重複しないための番号となる
      iv. 21〜25 は重複があるなら「num + 1」である
      v. 28 はファイルのコピーをしている
      vi. 30 はファイルのコピーが終了したらプログラムを終了するとういことである

(e) 33 はファイルが見つからない場合に読み込まれる

3. 実行結果

(a) 01,02 より初めは 4 つのファイルしか存在してないことがわかる

(b) 03〜08 では text.sh をコピーしている

(c) 06,07 より重複しない為に数字が増えていることがわかる

(d) 09,10 では存在しないファイル名

6.4.2 利点・欠点

1. 利点

(a) 複製が早くなった

(b) 重複しないファイル名がすぐに出せる

2. 欠点

(a) 拡張子を入力しないといけない

(b) ファイル名-10 以降が作れない

i. 実行結果

```
01  oshiaki-oshiro-no-macbook-2:b yoshiaki$ ./LevelX-1.sh text.sh sh
02  ファイル名 [ text-8.sh ] でコピーしました
03  yoshiaki-oshiro-no-macbook-2:b yoshiaki$ ./LevelX-1.sh text.sh sh
04  ファイル名 [ text-9.sh ] でコピーしました
05  yoshiaki-oshiro-no-macbook-2:b yoshiaki$ ./LevelX-1.sh text.sh sh
06  ファイル名 [ text-10.sh ] でコピーしました
07  yoshiaki-oshiro-no-macbook-2:b yoshiaki$ ./LevelX-1.sh text.sh sh
08  ファイル名 [ text-10.sh ] でコピーしました
09  yoshiaki-oshiro-no-macbook-2:b yoshiaki$ ./LevelX-1.sh text.sh sh
10  ファイル名 [ text-10.sh ] でコピーしました
```

ii. 原因はファイルの読み込む順番以下の通りになりるからである

実行結果（予想）			実行結果（実際）		
初期値	num =1		初期値	num =1	
text-1.sh = text-1.sh	num =2	○	text-1.sh = text-1.sh	num =2	○
text-2.sh = text-2.sh	num =3	○	text-10.sh = text-2.sh	num =2	×
text-3.sh = text-3.sh	num =4	○	text-2.sh = text-2.sh	num =3	○
text-4.sh = text-4.sh	num =5	○	text-3.sh = text-3.sh	num =4	○
text-5.sh = text-5.sh	num =6	○	text-4.sh = text-4.sh	num =5	○
text-6.sh = text-6.sh	num =7	○	text-5.sh = text-5.sh	num =6	○
text-7.sh = text-7.sh	num =8	○	text-6.sh = text-6.sh	num =7	○
text-8.sh = text-8.sh	num =9	○	text-7.sh = text-7.sh	num =8	○
text-9.sh = text-9.sh	num =10	○	text-8.sh = text-8.sh	num =9	○
text-10.sh = text-10.sh	num =11	○	text-9.sh = text-9.sh	num =10	○
text.sh = text-10.sh	num =11	×	text.sh = text-10.sh	num =10	×

※○は同じファイル名なので num の値は「＋ 1 」

※×は違うファイル名なのでそのまま

iii. 読み込む順番が「text-9.sh」よりも「text-10.sh」が早いのが原因である

6.5 スクリプト本体 (LevelX-2.sh)

1. ファイル名だけ入力てコピーするプログムの作成した
2. 作成出来るファイル数を 10 個から 100 個に増やした
3. 作成したスクリプト LevelX-2.sh を以下に示す。
4. 変更した部分だけ示す

```
01  #!/bin/sh
02  #————————スクリプトに渡された引数の数を数える————————-#
03  kazu=$#
04  if [ $kazu -eq 1 ] ; then
05  ext1=$1
06  else
07      echo ”引数 1 個必要（ファイル名）”
08      exit 1
09  fi
10  #————————現在の場所のファイル名を取得————————-#
11  filelist=‘/bin/ls‘
12  #————————指定したファイルの拡張子の取得————————-#
13  ext2=‘echo ext1   cut -d”.” -f2—‘

18        #————————拡張子を取り除いた文字列を求める————————-#
19        name=‘echo $ext1   cut -d”.” -f1—‘
20        num=01
21        #————————ファイル名に重複がないか判断————————#
22        for file in $filelist
23        do
24          if [ $file = $name-$num.$ext2 ] ; then
25            num=0‘expr $num + 1‘
26            #————————先頭に 0 をつけないパターン————————#
27            if [ $num -gt 9 ] ; then
28              num=‘expr $num‘
29            fi
30          fi
31        done
```

6.6 実行結果 (LevelX-1.sh)

LevelX-1.sh を実行した結果を示す.

```
01  yoshiaki-oshiro-no-macbook-2:b yoshiaki$ ls
02  LevelX-1-1.sh LevelX-1.sh  LevelX-2.sh  test.sh
03  LevelX-1.sh LevelX-2.sh test.sh

04  yoshiaki-oshiro-no-macbook-2:b yoshiaki$ ./LevelX-2.sh test.sh
05  ファイル名 [ test-01.sh ] でコピーしました

    < 省略 >

06  yoshiaki-oshiro-no-macbook-2:b yoshiaki$ ./LevelX-2.sh test.sh
07  ファイル名 [ test-10.sh ] でコピーしました
08  yoshiaki-oshiro-no-macbook-2:b yoshiaki$ ./LevelX-2.sh test.sh
09  ファイル名 [ test-11.sh ] でコピーしました

10  yoshiaki-oshiro-no-macbook-2:b yoshiaki$ ./LevelX-2.sh test
11  ファイルが存在しません

12  yoshiaki-oshiro-no-macbook-2:b yoshiaki$ ./LevelX-2.sh
13  引数 1 個必要（ファイル名）

14  yoshiaki-oshiro-no-macbook-2:b yoshiaki$ ls
15  LevelX-1-1.sh    LevelX-2.sh    test-04.sh    test-08.sh    test.sh
16  LevelX-1.sh      test-01.sh     test-05.sh    test-09.sh    test.sh
17  LevelX-1.sh~     test-02.sh     test-06.sh    test-10.sh
18  LevelX-2.sh      test-03.sh     test-07.sh    test-11.sh
```

6.7 考察 (LevelX-1.sh)

6.7.1 説明

1. 流れ（./LevelX-2.sh test）
   (a) 入力された値が 1 つかどうかの判断（1 つ表示されているので 05）
   (b) 全てのファイル名を取得し、入力されたファイルの拡張子を取得する
   (c) 同じファイルが存在するか判断（14〜32 の for 文）
   (d) 新しいファイル名を作成し、重複がないかの判断を行う
   (e) ここで重複がある場合は新しいファイル名を作成する
   (f) 「1〜9」の時は「01〜09」のように頭に 0 をつける
   (g) 二桁の時はそのまま表示する
   (h) もう一度判断させる
   (i) それを延々と続けることにより、いずれファイル名が重複しないファイル名が作成される
2. スクリプト本体

(a) 05 より ext1 だけにした

(b) 07 の出力する文字を変更した

(c) 13 ではファイル名から拡張子だけを取得している

(d) 「-f2」より「 . 」の左側を取得するようにした

(e) 20 より「01」に変更した

(f) 25 より「0(数字)」で設定した

(g) 27〜29 は num が 10 以上のとき「(数字)」で出力。つまり、先頭に 0 をつけない

3. 実行結果

(a) 01〜03 は実行する前の状態である

(b) 04 より「text.sh」のコピーを作成する

(c) 05 より「text-01.sh」作成出来たと言っている

(d) 09 より 11 以降も出来ていることがわかる

(e) 10 では「test」のように拡張子を入れないでやってみた

(f) 11 よりちゃんと処理が出来ていることがわかる

(g) 12 ではファイル名を書かないでやってみた

(h) 13 よりちゃんと処理が出来ていることがわかる

(i) 14〜16 より「test-11.sh」などちゃんと作成されていることがわかる

7　参考文献

1. http://www002.upp.so-net.ne.jp/latex/index.html（LaTeX コマンドシート一覧）
2. `http://www.cityfujisawa.ne.jp/~huzinami/tex/color.html`（LaTeX　色の付け方）
3. http://www.pythonweb.jp/tutorial/index.html (Python 入門)
4. http://omake.accense.com/wiki/Python（Python 情報）
5. http://d.hatena.ne.jp/daijiroc/20090207/1233980551（echo コマンドの出力の色を変更する）
6. http://www.geocities.jp/geo_sunisland/filter_etc.html#flt-etc-1-2 (cut コマンドを使用したフィルタリング)
7. http://t16web.lanl.gov/Kawano/gnuplot/ （gnuplot 入門）