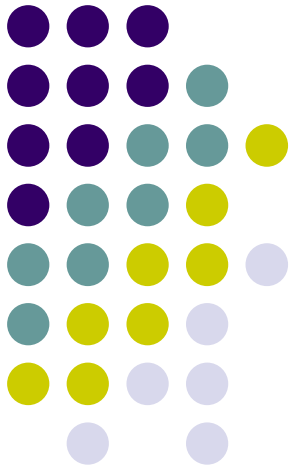
# 第2回　BASHゼミ



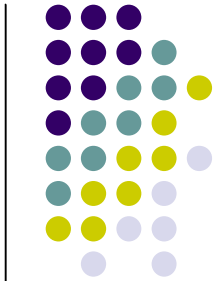
- 値の代入と参照
- クォート
- 環境変数、パラメータ変数
- 配列、宣言
- 局所変数

発表者：鈴木　朋央

# 値の代入と参照(1/2)

- 変数は宣言せずに使用できる
  - ・変数xに文字列を代入する場合
  - → x=hoge
  - ＊等号の前後にスペースを入れてはいけない
  - ・xを参照する場合
  - → echo $x
  - ＊変数xの前に「$」をつける
  - ＊$xは${x}としてもよい

# 値の代入と参照(2/2)



- 変数の値は文字列型として扱われる
- 数値の操作をするときに数値文字列を数値に変換する
- 大文字と小文字は区別される.変数の開放にはunsetを使用する.

  例：$ x=100

  $ echo $x

  100

  $ unset x

  $ echo $x

# クォート～ダブルクォーテーション

- パラメータは空白,タブ,改行コードで区切られる
- 変数にこれらの値を代入するには,その値をダブルクォーテーションで囲う

  → x="hoge foo"

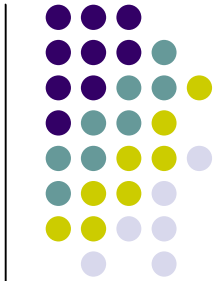
- $変数名を囲った場合、その変数は展開される

  → echo $x

  echo “$x”

  ＊これらの結果は等しい

# クォート～シングルクォーテーション



- $変数名を囲った場合、その変数は展開されない

  → echo ‘$x’

  ＊結果は$xとなる

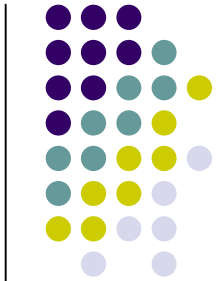
- 囲った部分をBASHに解釈させたくない場合に使う

  使用例：xで始まる全パッケージをインストール

  → echo -e '\a'

  ＊\は特殊文字であるため、囲う必要がある

# クォート～エスケープシーケンス



- \$変数名だと,\が$の持つ特別な意味を書き消す

  → echo \$x

  ＊結果は$xとなる

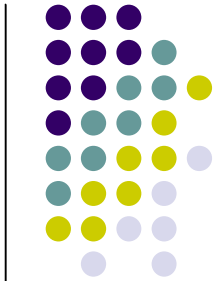
- \もBASHに解釈させたくない特殊文字を使用するときに使う

  → echo –e ‘\a’

  echo –e \\a

  ＊これらの結果は等しい

# 環境変数



- あらかじめ環境で設定してある値で初期化される変数

  例：$HOME　ホームディレクトリのパス

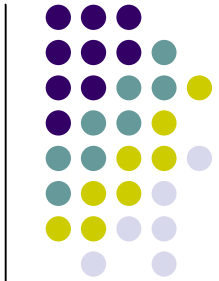
  　　$OS　使用されているOS　　・・・etc

- 普通大文字である

  → ユーザ定義の変数は小文字のほうがよい？

- コマンドenvで確認できる

# パラメータ変数(1/3)



- $0 スクリプトの名前
- $# 渡されたパラメータの数
- $$ スクリプトのプロセスID
- $1,$2,... スクリプトのパラメータ
- $* 全パラメータを表す一つの変数.各パラメータは$IFSで区切られる.ダブルクォートしないと,BASHに$1,$2,....と展開される気がする(*は全ての文字列に一致する).
- $@ 全パラメータを表す一つの変数.各パラメータは$IFSに関係なく(おそらくスペースで)区切られる.

# パラメータ変数(2/3)

- $*

  $IFSの最初の文字で区切って並べた1つの単語に展開される

  → “**$***” は “**$1***c***$2***c***...**” と等しい

  (cは$IFSの最初の文字)

- $@

  それぞれのパラメータは別々の単語に展開されます

  → “**$@**” は “**$1**” “**$2**” … と等しい

# パラメータ変数(3/3)

- 例：./sample10.sh ａ ｂ ｃ ｄを実行した場合

$* = ａ ｂ ｃ ｄ　・・・引数リスト
$@ = ａ ｂ ｃ ｄ　・・・引数リスト
$# = 4　・・・・・・引数の数
$0 = sample01.sh・・コマンド
$1 = ａ　・・・・・・第一引数
$2 = ｂ　・・・・・・第二引数
$3 = ｃ　・・・・・・第三引数
$4 = ｄ　・・・・・・第四引数

# 配列(1/5)

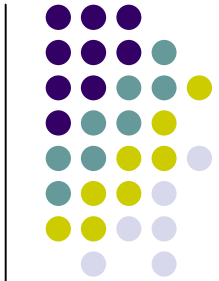
- 1次元配列の使用が可能
- 通常の変数と同様に宣言なしで使用できる
- 配列の場合はブラケット{ }が常に必須である

  例外として・・・

  ・0番目の要素を指定するときに限り、ブラケット、インデクスは使用しなくてもよい。

  ・ブラケットを使用してインデクスを指定しない場合も0番目の値を返す.
- 0番目の要素を初期化していない場合、空を返す

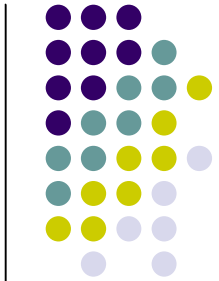
# 配列(2/5)



```
$ ARR[0]=1
$ echo ${ARR[0]}
1
$ echo $ARR
1
$ echo ${ARR}
1
----------------------------------
$ ARR[100]=100
$ echo ${ARR[100]}
100
```

```
$ B[3]=3
$ echo $B
＊空表示
$ echo ${B}
＊空表示
$ echo ${B[3]}
3
```

# 配列(3/5)



- 配列への値の代入には,複合代入も使用できる

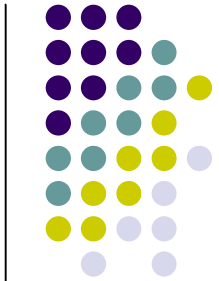
  → 配列名=(値1 値2 値3 値4 値5....)
- 値は0番目の要素から順に格納されていく
- インデクスに*か@を指定すると,その配列の全要素が返る

  $ C=(1 2 3 4 5)

  $ echo ${C[*]}

  1 2 3 4 5

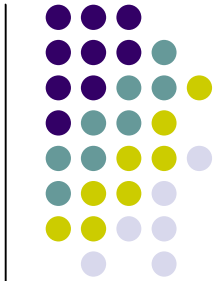
# 配列(4/5)



- 配列名の頭に#をつけ,インデクスに*か@を指定すると,配列の要素数が返る

```
$ echo ${#C[*]}
5
$ echo ${#C[@]}
5
```

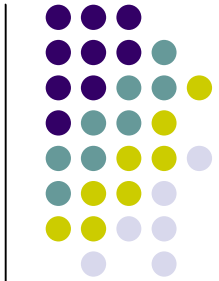
# 配列5/5)



- 配列の領域を開放するには,通常の変数と同様にunsetを使用する

```
$ echo ${B[@]}
4 3
$ unset B
$ echo ${B[@]}
＊空表示
```

# 宣言の概要



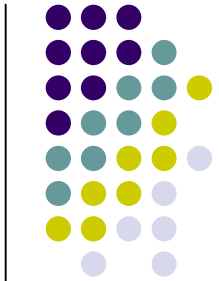
- 変数を明示的に宣言することもできる
- 変数に対していくつかの属性を付加することができる
- 宣言にはdeclareあるいはtypesetを使用する

  declare [-afFirx] [-p] [name[=value]]

  typeset [-afFirx] [-p] [name[=value]]

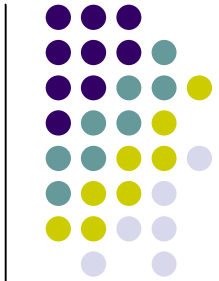
  ＊name：変数名　value：初期化値
- nameを指定しなければ,現在の全変数の値を表示する

# 宣言～オプションの詳細



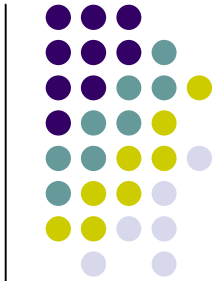
- a：変数を配列変数として宣言する.
- f ：変数を関数として扱う.
- F ：関数定義の表示を抑制し,関数名と属性のみを表示する.-Fを指定すると-fも指定したことになる.
- i ：変数を整数として扱う.変数に値が代入されたときに,算術式を評価するようになる.
- r ：変数を読み込み専用変数として扱う.読み込み専用変数には値を代入したりunsetできなくなる(※).
- x：変数をエクスポートする.
- p：変数の属性と値を表示する.

# 宣言～typeset使用例（配列）



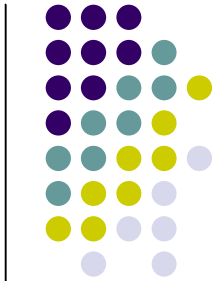
```
$ typeset -a ARR
$ typeset -p ARR
declare -a ARR='()'
$ ARR=(1 2 3 4 5)
$ typeset -p ARR
declare -a ARR='([0]="1" [1]="2" [2]="3" [3]="4"
   [4]="5")'
$ unset ARR
$ typeset -p ARR
bash: typeset: ARR: not found
```

# 宣言～declare使用例 (関数格納)



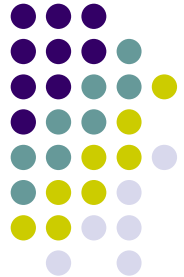
```
$ declare -f func
$ function func1 () { echo 'This is func1'; }
$ function func2 () { echo 'This is func2'; }
$ func=func1
$ $func
This is func1
$ func=func2
$ $func
This is func2
```

# 宣言～読み込み変数の使用例



```
$ typeset -r RO
$ RO=1
bash: RO: readonly variable
$ unset RO
bash: unset: RO: cannot unset: readonly variable
$ typeset +r RO
bash: typeset: RO: readonly variable
$ typeset -r RO=1
bash: typeset: RO: readonly variable
```

# 宣言～読み込み変数の特性



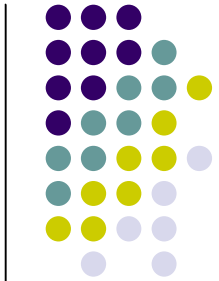
読み込み専用変数は読み込み専用属性を取り
除くことも上書き的に宣言し直すこともできない

↓

値を持たせたいときには,typesetやdeclare時に
忘れずに値を指定する ！！

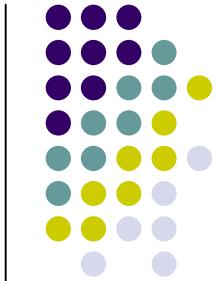
※ readonlyというコマンドも同様の機能を提供する

# 局所変数(1/3)



- 特に指定しない限り変数は全て大域変数である
- 変数を局所変数として指定するには,以下の方法がある
  - 関数内でdeclareやtypesetコマンドを使用して,変数を宣言する.
  - 関数内でlocalコマンドを使用して変数を宣言する.
- いずれもスコープは局所変数を宣言した関数とその子の関数である

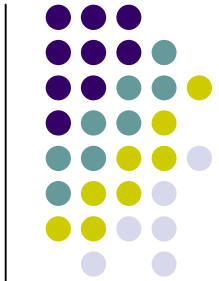
# 局所変数(2/3)



- localコマンドはdeclare,typesetを局所変数に限定したようなもの
- 両者と同じ引数をとるが,関数内以外で使用するとエラーとなる

```
$ local LOCAL
bash: local: can only be used in a function
```

# 局所変数(3/3)

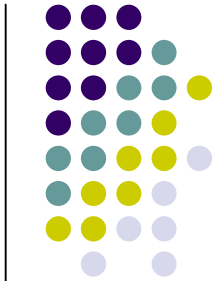


```
$ typeset -i GRB=0
$ function localTest() {
        local -i LOCAL;
        let LOCAL++;
        let GRB++;
        echo
   "LOCAL=$LOCAL";
        echo "GRB=$GRB";
}
$ localTest
$ localTest
```

＊実行結果

```
LOCAL=1
GRB=1
LOCAL=1
GRB=2
```

# 練習問題1



- コマンドラインから2つの整数を引数として渡し、それらを局所変数として四則演算を行った結果と、渡された変数を表示する関数を作成しなさい。

  ＊四則演算は let コマンドを用いる