C2110 *UNIX and programming*

Lesson 10 / Module 2

PS / 2020 Distance form of teaching: Rev5

Petr Kulhanek

kulhanek@chemi.muni.cz

National Center for Biomolecular Research, Faculty of Science
Masaryk University, Kamenice 5, CZ-62500 Brno

Obsah

- **AWK**
  - **What is the AWK language for?**
  - **Script structure, course of execution**
  - **Block structure, regular expressions, script execution**
  - **Variables, operations on variables**
  - **Formatted and unformatted output**

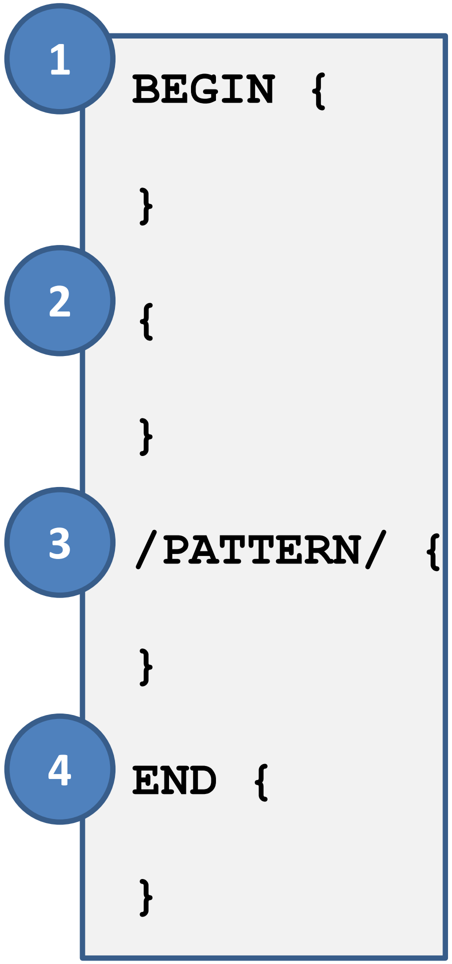
AWK

http://www.gnu.org/software/gawk/gawk.html

AWK is a scripting language designed for **text data processing**, whether in the form of text files or streams. The language uses **string data types**, **associative field** (arrays indexed by string keys) and **regular expressions**.

adapted from www.wikipedia.org

Process of Executing Script

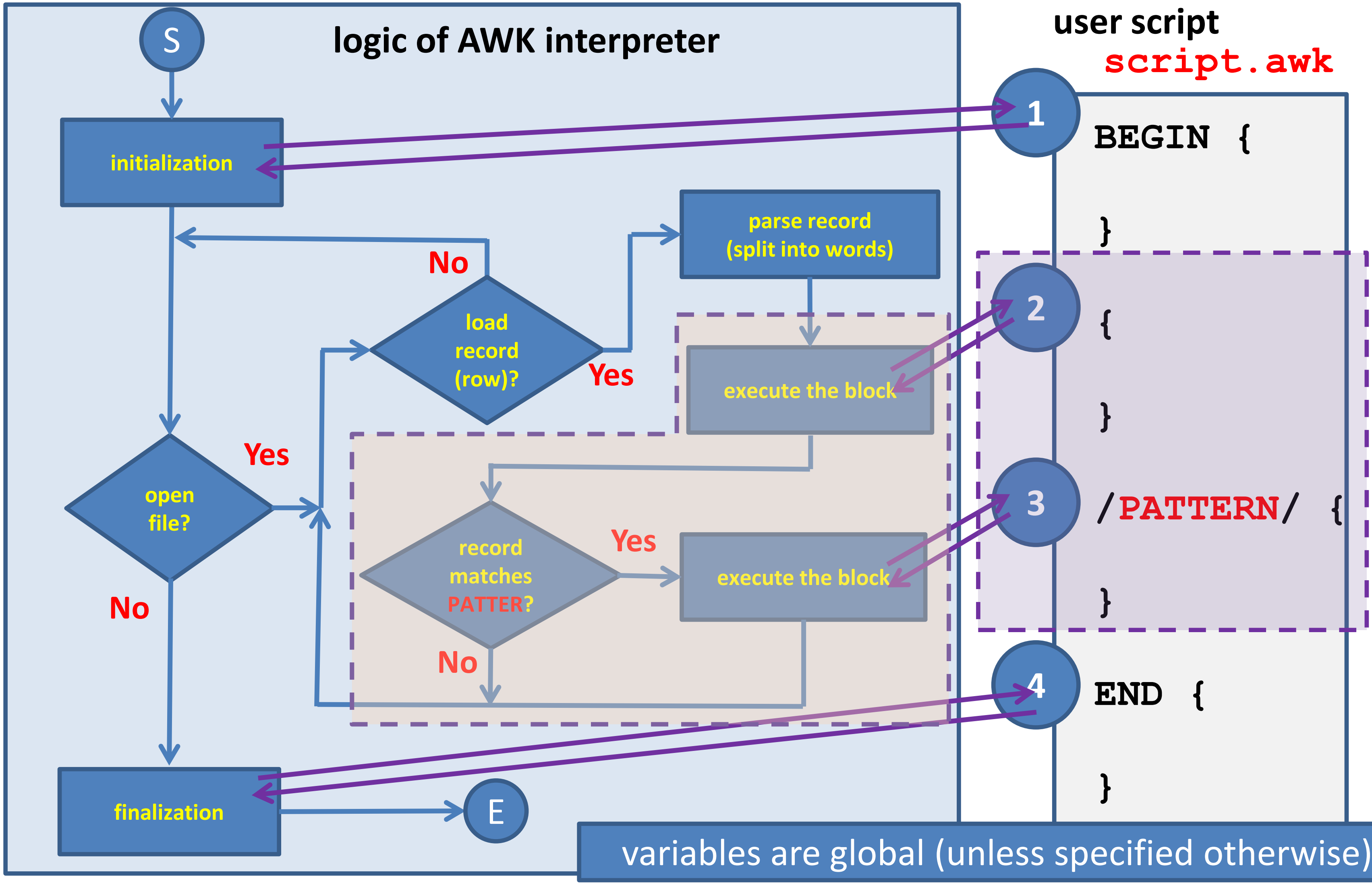


```
BEGIN {

}

{

}

/PATTERN/ {

}

END {

}
```

- BEGIN (1) block is executed (if included in the script) before parsing the file.
    - The record is loaded from the file. By default, the record is the entire line of the analyzed file or stream. The record is divided into fields. By default, the fields are individual words in the record.
    - Block (2) is executed for the given record.
    - If the record matches PATTERN, block (3) is executed.
    - .... possibly other blocks are executed ....
- END (4) block is executed (if included in the script) after parsing the entire file.

Process of Executing Script

```
awk -f script.awk file1.txt file2.txt
```



```
BEGIN {

}

{

}

/PATTERN/ {

}

END {

}
```

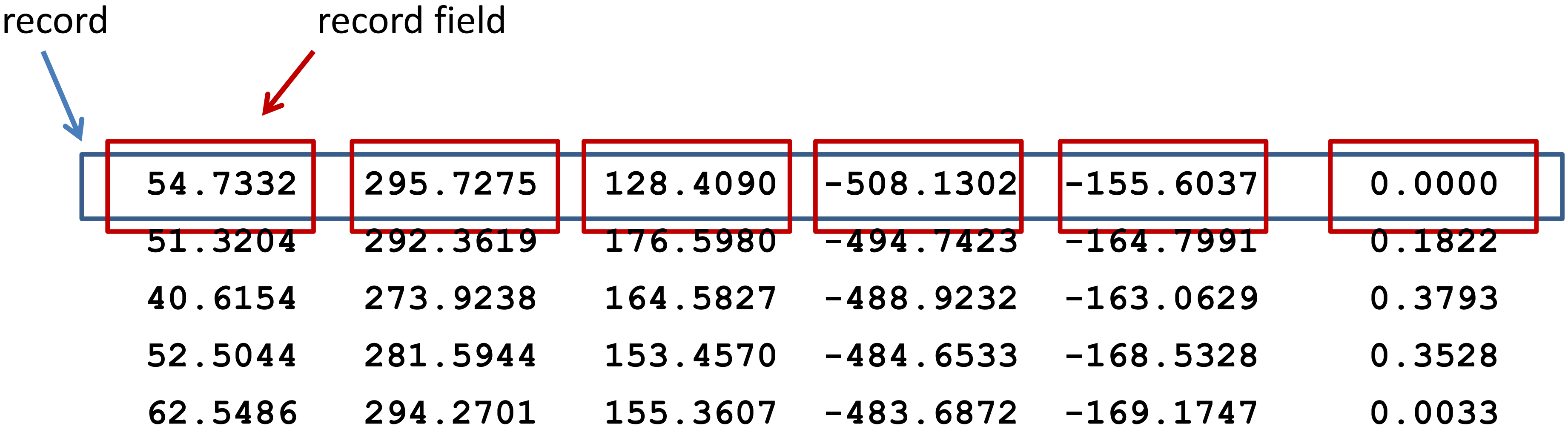
variables are global (unless specified otherwise)

Analysis of Text Files

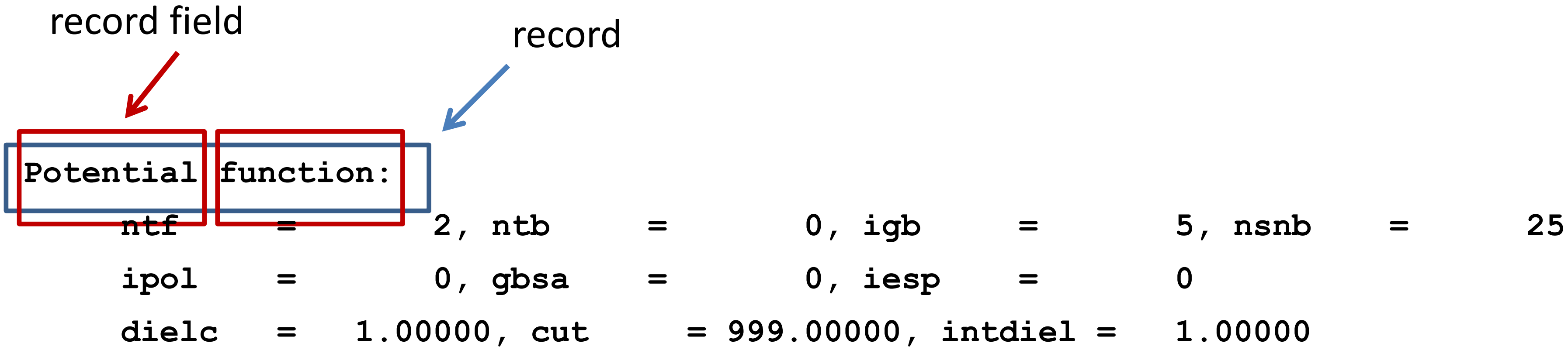
```
54.7332   295.7275   128.4090  -508.1302  -155.6037     0.0000
51.3204   292.3619   176.5980  -494.7423  -164.7991     0.1822
40.6154   273.9238   164.5827  -488.9232  -163.0629     0.3793
52.5044   281.5944   153.4570  -484.6533  -168.5328     0.3528
62.5486   294.2701   155.3607  -483.6872  -169.1747     0.0033
```

```
Potential function:

     ntf     =       2, ntb     =       0, igb     =       5, nsnb    =      25
     ipol    =       0, gbsa    =       0, iesp    =       0
     dielc   =   1.00000, cut     = 999.00000, intdiel =   1.00000
```

Analysis of Text Files

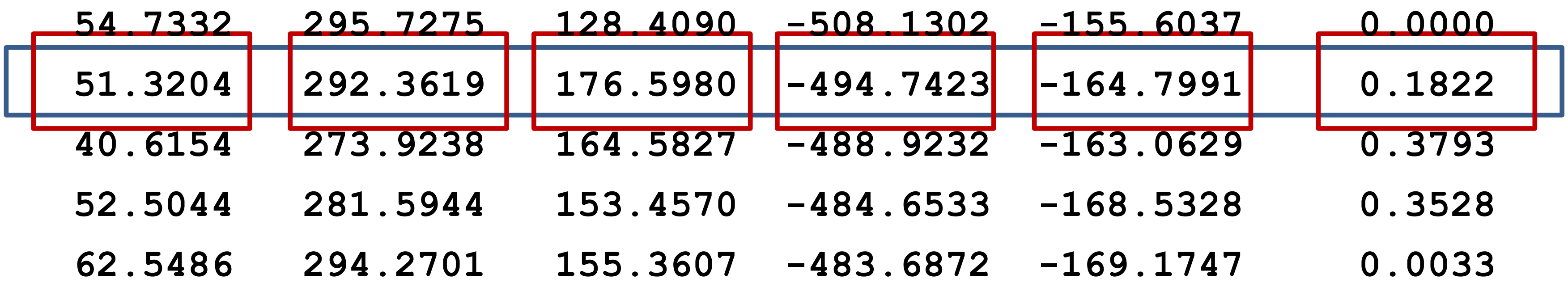


```
  54.7332   295.7275   128.4090  -508.1302  -155.6037     0.0000
  51.3204   292.3619   176.5980  -494.7423  -164.7991     0.1822
  40.6154   273.9238   164.5827  -488.9232  -163.0629     0.3793
  52.5044   281.5944   153.4570  -484.6533  -168.5328     0.3528
  62.5486   294.2701   155.3607  -483.6872  -169.1747     0.0033
```

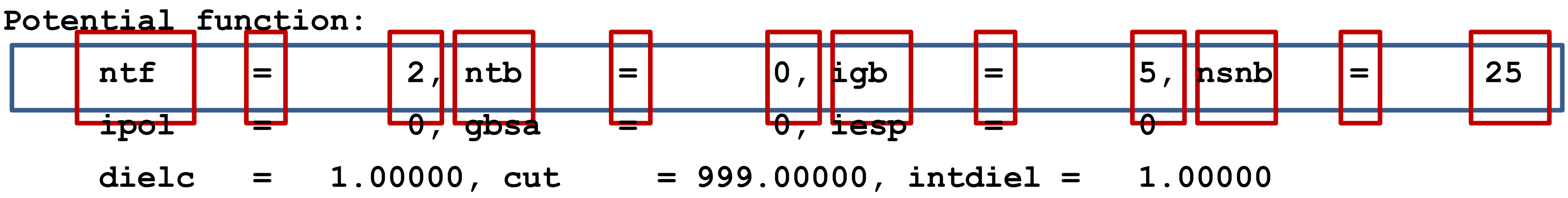


```
Potential function:
    ntf     =       2, ntb     =       0, igb     =       5, nsnb    =      25
    ipol    =       0, gbsa    =       0, iesp    =       0
    dielc   =   1.00000, cut     = 999.00000, intdiel =   1.00000
```

Analysis of Text Files



```
   54.7332    295.7275    128.4090   -508.1302   -155.6037      0.0000
   51.3204    292.3619    176.5980   -494.7423   -164.7991      0.1822
   40.6154    273.9238    164.5827   -488.9232   -163.0629      0.3793
   52.5044    281.5944    153.4570   -484.6533   -168.5328      0.3528
   62.5486    294.2701    155.3607   -483.6872   -169.1747      0.0033
```



```
Potential function:
    ntf     =       2, ntb      =       0, igb     =       5, nsnb    =      25
    ipol    =       0, gbsa     =       0, iesp    =       0
    dielc   =   1.00000, cut      = 999.00000, intdiel =   1.00000
```

Example

vstup.txt

```
  54.7332   295.7275   128.4090   -508.1302   -155.6037     0.0000
  51.3204   292.3619   176.5980   -494.7423   -164.7991     0.1822
  40.6154   273.9238   164.5827   -488.9232   -163.0629     0.3793
  52.5044   281.5944   153.4570   -484.6533   -168.5328     0.3528
  62.5486   294.2701   155.3607   -483.6872   -169.1747     0.0033
```

script.awk

```
{
 print $ 2;
}
```

one simple block

```
$ awk -f script.awk input.txt
```

nebo

```
$ awk '{ print $2; }' input.txt
```

```
295.7275
292.3619
273.9238
281.5944
294.2701
```

Block Structure, Example

```
# block calculates running sum of second column
# and running sum of fourth column if third column
# contains value 5
{
   # this is a comment
   f = f + $2; # here I calculate running sum
   printf(“Running sum is %10.3f\n",f);
   if( $3 == 5 ) {
      k = k + $4; # running sum for fourth column
   }
}
# block for cumulative sum of temperature (fifth column)
# on lines containing keyword "TEMP"
/TEMP/ {
   temp = temp + $5;
}
```

- comments are preceded by a # character
- commands are presented on separate lines, which should end with a semicolon
- a semicolon must be used if we specify two or more commands per line

PATTERN - Regular Expressions

```
/PATTERN/ {

}
```

If PATTERN matches the record, the block is executed.

The pattern is **regular expression**.

Regular expression is a language that describes the structure of a text string. The language is used to search for text strings, to replace part of strings.

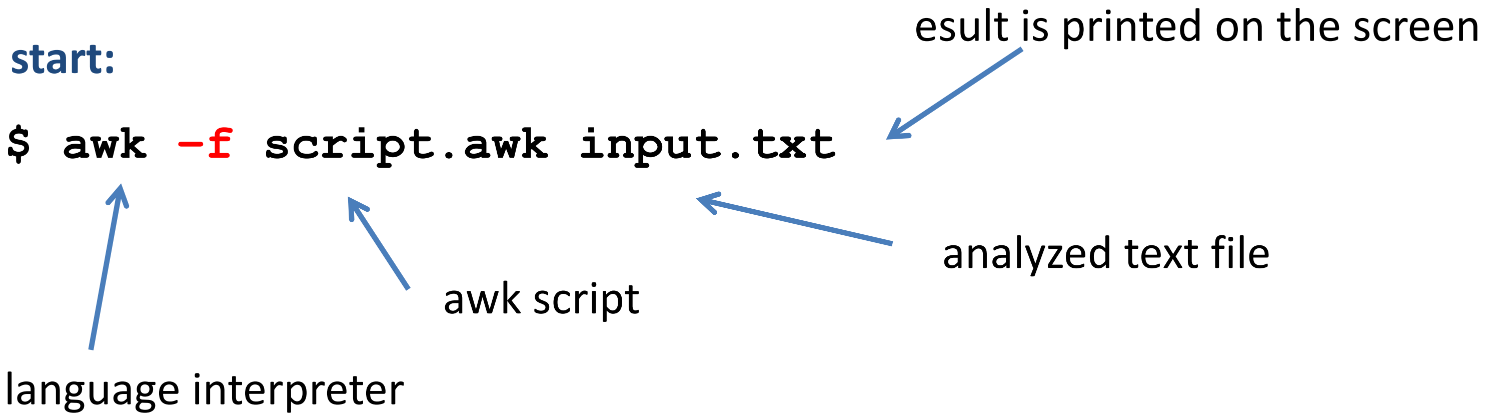
Examples of simple regular expressions:

TEXT - is met if the record contains TEXT (can be anywhere)
^TEXT - is met if the record contains TEXT at the beginning
TEXT$ - is met if the record contains TEXT at the end

Starting AWK scripts

Text file processing:

Indirect start:



```
$ awk -f script.awk input.txt
```

esult is printed on the screen

analyzed text file

awk script

language interpreter

The analyzed data can be sent via standard input:

```
$ awk -f script.awk < input.txt

$ cat file.txt | awk -f script.awk
```

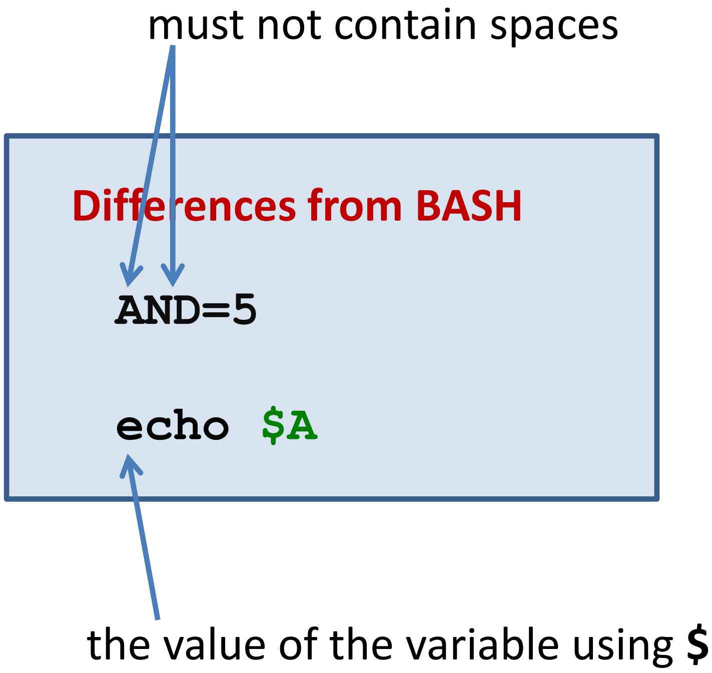
Exercise 1

1. Create a directory awk-data.
2. Copy the files matice.txt, produkt.log, and rst.out from directory /home/kulhanek/Documents/C2110/Lesson10 into directory awk-data .
3. Write a script that prints the second column from the matrix.txt file.
4. Write a script that prints the second and fourth column of the matice.txt file.

Variables

Assignment to a variable:

```
A = 10;
B = "this is a text "
C = 10.4567;
D = A + C;
```



must not contain spaces

Differences from BASH

```
AND=5

echo $A
```

the value of the variable using **$**

Variable value:

```
print A + C;
print B;
```

Special variables:

NF number of fields in the current record (Number of Fields)

NR order of record being processed (Number of Records)

FS field delimiter in record (Field Separator), **default is space and tab**

RS record separator (Record Separator), **default is newline character \n**

$ 0 whole record

$1, $2, $3 ... individual record fields

Variables, ...

$0	whole record
$1, $2, $3 ...	individual record fields

character **$** allows programmatic access to individual fields of the record

Example:

```
i = 3;
print $i;
```

prints value of the field specified by the value of the variable *i*

Mathematical operations

If a variable can be interpreted as a number, following arithmetic operators can be used:

++ increases the value of the variable by one

```
A++;
```

– decreases the value of the variable by one

```
A–;
```

+ sereads two values

```
A = 5 + 6;
A = A + 1;
```

- subtracts two values

```
A = 5 - 6;
A = A - 1;
```

***** multiplies two values

```
A = 5 * 6;
A = A * 1;
```

/ divide by two values

```
A = 5 / 6;
A = A / 1;
```

+= adds a value to the variable

```
A += 3;
A += B;
```

-= subtracts value from variable

```
A -= 3;
A -= B;
```

***=** multiplies variable by value

```
A *= 3;
A *= B;
```

/= divides variable by value

```
A /= 3;
A /= B;
```

Command print

Command **print** is used for unformatted printing of strings and numbers.

Syntax:

```
print value1[,] value2[,] ...;
```

if two values are separated by a comma, values in the output are separated by a space

Examples:

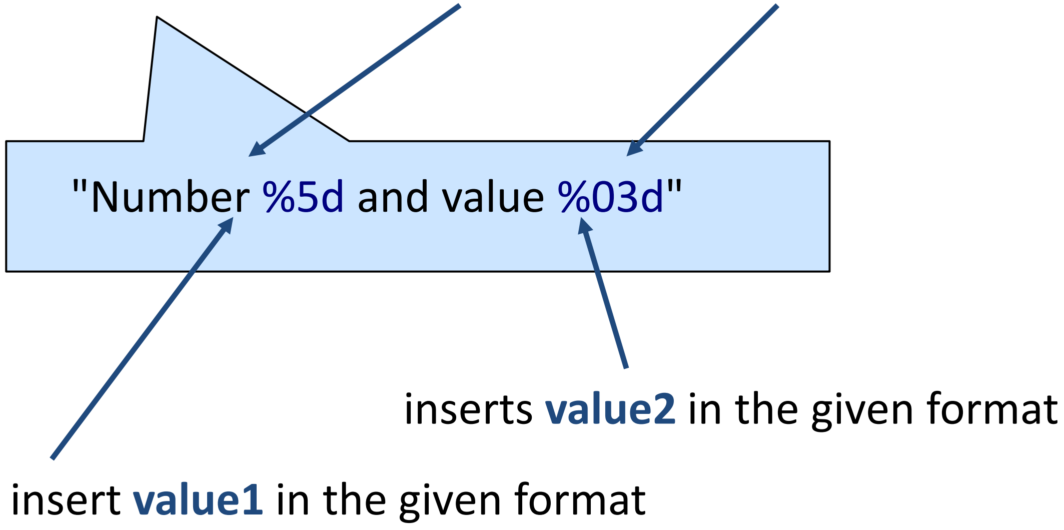
```
i = 5;
k = 10.456;
j = "value of variable i = ";
print j, i;
print "value variable k = ", k;
```

Function printf

Function **printf** is used for formatted printing of texts and numbers.

Syntax:

```
printf("format", value1, value2, ...);
```



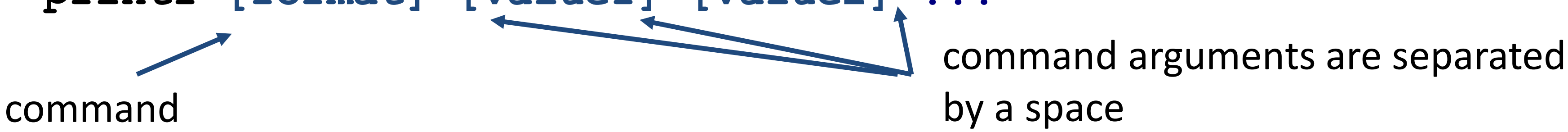
"Number %5d and value %03d"

inserts **value2** in the given format

insert **value1** in the given format

Difference to BASH:

```
printf [format] [value1] [value2] ...
```



command

command arguments are separated by a space

Exercise 2

1. Write a script that sums numbers in the second column of the matice.txt file.
2. Write a script that prints the number of lines that the matice.txt file contains. Verify the result with the command wc.

Self-study

Starting AWK scripts, ...

Direct start

```
$ ./script.awk input.txt

$ ./script.awk < input.txt

$ cat file.txt | ./script.awk
```

Script `script.awk` **must** have the x flag (**executable**) and AWK interpreter set (part of script).

```
#!/usr/bin/awk -f
{
    i += NF;
}
END {
    print “Number of words:", i;
}
```

I do not recommend using this method of starting AWK.