Awk:

Hack the planet['s text]!

Benjamin Porter

Benjamin Porter

- Software & Infrastructure Engineer at SimpleNexus
- Been programming for 15 years
- Fedora Linux user, lover of open source

- Twitter: @Freedom_Ben
- Keybase: FreedomBen - https://keybase.io/freedomben
- Email: FreedomBen@protonmail.com

Outline

- Why learn awk?
- What is awk?
- History of awk
- Super simple awk programs
- Awk Patterns Overview
- Awk Actions Overview
- Dive a Little Deeper (functions, pipes)
- Example programs

Why Learn Awk?

This is an excellent question! There are several good reasons:

- Awk is part of Posix, so it is installed *everywhere*
- Many of the problems you face are text processing problems
- Awk is the gold standard of text processing tools
- People are impressed and enamored with people that use awk
- Awk will make you powerful
- All *real* hackers use awk
- You want to be cool, don’t you?

What is Awk?

- A powerful, succinct scripting language for text processing
- More formally, Awk is a data-driven scripting language consisting of a set of actions to be taken against streams of textual data for purposes of extracting or transforming text, such as producing formatted reports
- Written by Alfred Aho, Peter Weinberger, and Brian Kernighan
- Initially developed in 1977

- Source: https://en.wikipedia.org/wiki/AWK

What is Awk?

- Awk was significantly revised and expanded in 1985–88 into GNU Awk
- GNU Awk (gawk) written by Paul Rubin, Jay Fenlason, and Richard Stallman
- gawk is most widely deployed version
- gawk has been maintained solely by Arnold Robbins since 1994
- Brian Kernighan's nawk (New AWK) source was first released in 1993 unpublicized, and publicly since the late 1990s;
- Many BSD systems use nawk to avoid the GPL license (but their users always install gawk ;-))

- Source: https://en.wikipedia.org/wiki/AWK

Is awk a programming language?

- Awk is a command line tool, but more so than grep and others it is also a programming language!
- It’s not a general purpose language. It’s optimized for text processing
- But, it is Turing complete!

History of Awk?

Before Awk:

- Was preceded by sed, which was the scripting part of ed
- Sed was the first powerful regex tool
- Used main loop and current line variables (awk expanded on this)
- Awk was an evolution in the sed line-oriented approach

After Awk:

- Awk’s powerful regexes and also its limitations inspired Perl,
- Perl in turn inspired beautiful languages like Ruby which inspired Elixir
- We have a lot to thank awk for!

The Traditional “Hello World” in awk

- BEGIN { print "Hello, world!" }

Running an awk program

- `awk ‘program’ input files`
- `awk -f progfile` *`input files`*
- `some_command | awk ‘program’`

- `#!/usr/bin/env awk -f`
- `./script.awk *.log`

Structure of an awk program

- `pattern    { action }`

- Awk scans a sequence of input lines one after another searching for lines that are matched.
- Every input line is tested against each pattern in turn
- For each match, the { action } is executed
- After every applicable { action } is executed, the next line is processed
- Action are enclosed in braces to distinguish them from the pattern

Structure of an awk program

- Either the pattern or the action can be omitted
- If the pattern is omitted, every line will match

```
‘{ print $1 }’
```

- If the action is omitted, every matching line will be printed

```
‘/regex/`
```

Awk Patterns

- Awk patterns are basically just “if” statements to decide to execute the action
- Decide if a match is True or False
- If True, execute the following Action
- If False, skip the action and proceed to test the next pattern with current line

Summary of Patterns

1. `BEGIN` { *statements* }
   The *statements* are executed once before any input has been read.

2. `END` { *statements* }
   The *statements* are executed once after all input has been read.

3. *expression* { *statements* }
   The *statements* are executed at each input line where the *expression* is true, that is, nonzero or nonnull.

4. */regular expression/* { *statements* }
   The *statements* are executed at each input line that contains a string matched by the *regular expression.*

5. *compound pattern* { *statements* }
   A compound pattern combines expressions with `&&` (AND), `||` (OR), `!` (NOT), and parentheses; the *statements* are executed at each input line where the *compound pattern* is true.

6. *pattern*$_1$, *pattern*$_2$ { *statements* }
   A range pattern matches each input line from a line matched by *pattern*$_1$ to the next line matched by *pattern*$_2$, inclusive; the *statements* are executed at each matching line.

Awk Patterns

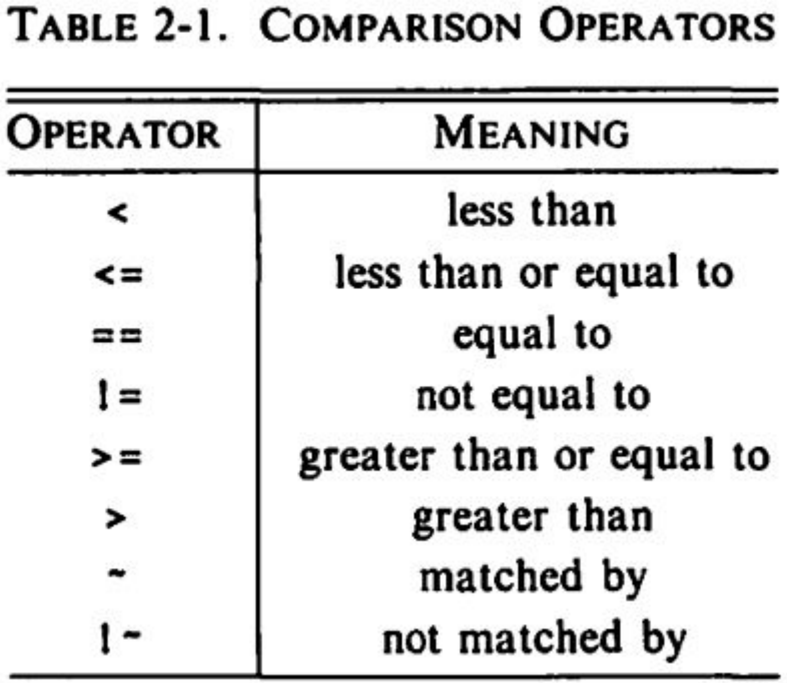


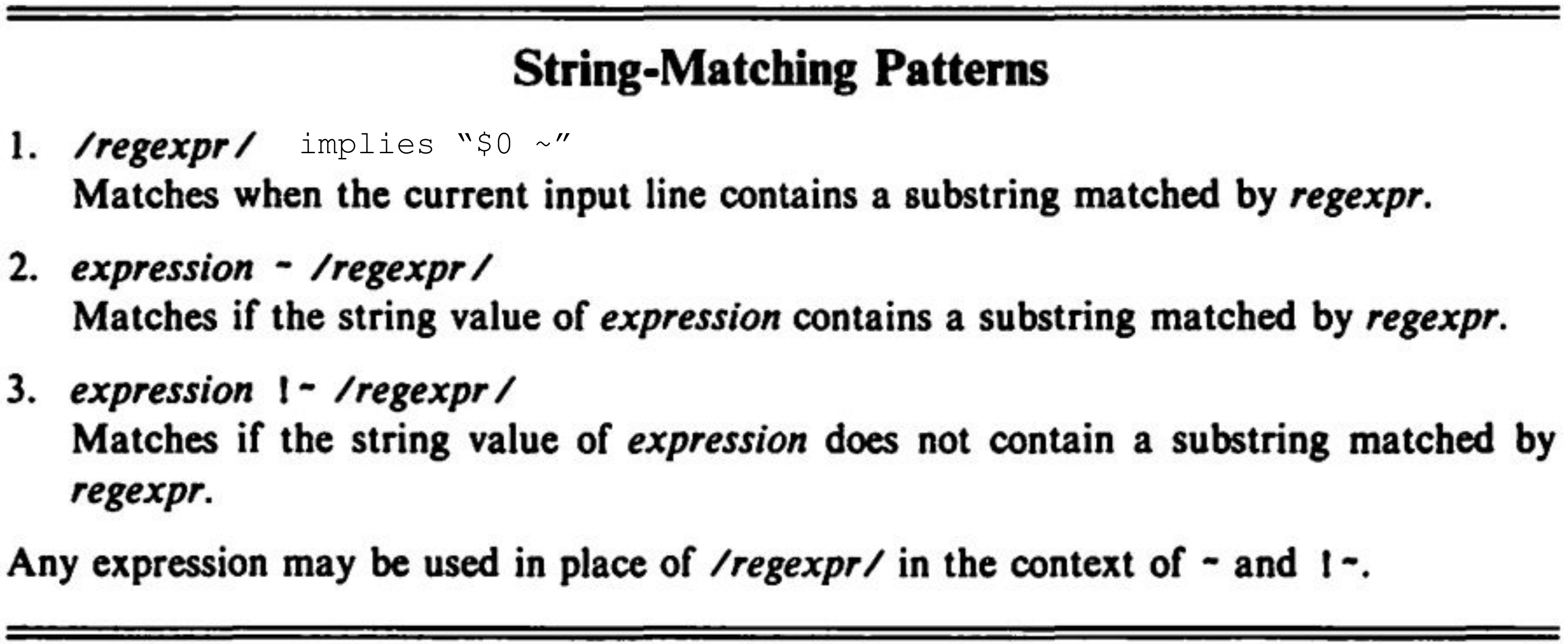
TABLE 2-1. COMPARISON OPERATORS

OPERATOR	MEANING
<	less than
<=	less than or equal to
==	equal to
!=	not equal to
>=	greater than or equal to
>	greater than
~	matched by
!~	not matched by

Examples:

```
NF < 10     # Num Fields

NR <= 150   # Num Records

$1 == "SomeString"

$4 ~ /linux/ (or "linux")

$5 !~ /awk/

$2/$3 >= 0.5
```

Awk Patterns



String-Matching Patterns

1. */regexpr/* `implies "$0 ~"`
   Matches when the current input line contains a substring matched by *regexpr*.

2. *expression* ~ */regexpr/*
   Matches if the string value of *expression* contains a substring matched by *regexpr*.

3. *expression* !~ */regexpr/*
   Matches if the string value of *expression* does not contain a substring matched by *regexpr*.

Any expression may be used in place of */regexpr/* in the context of ~ and !~.

Awk Patterns

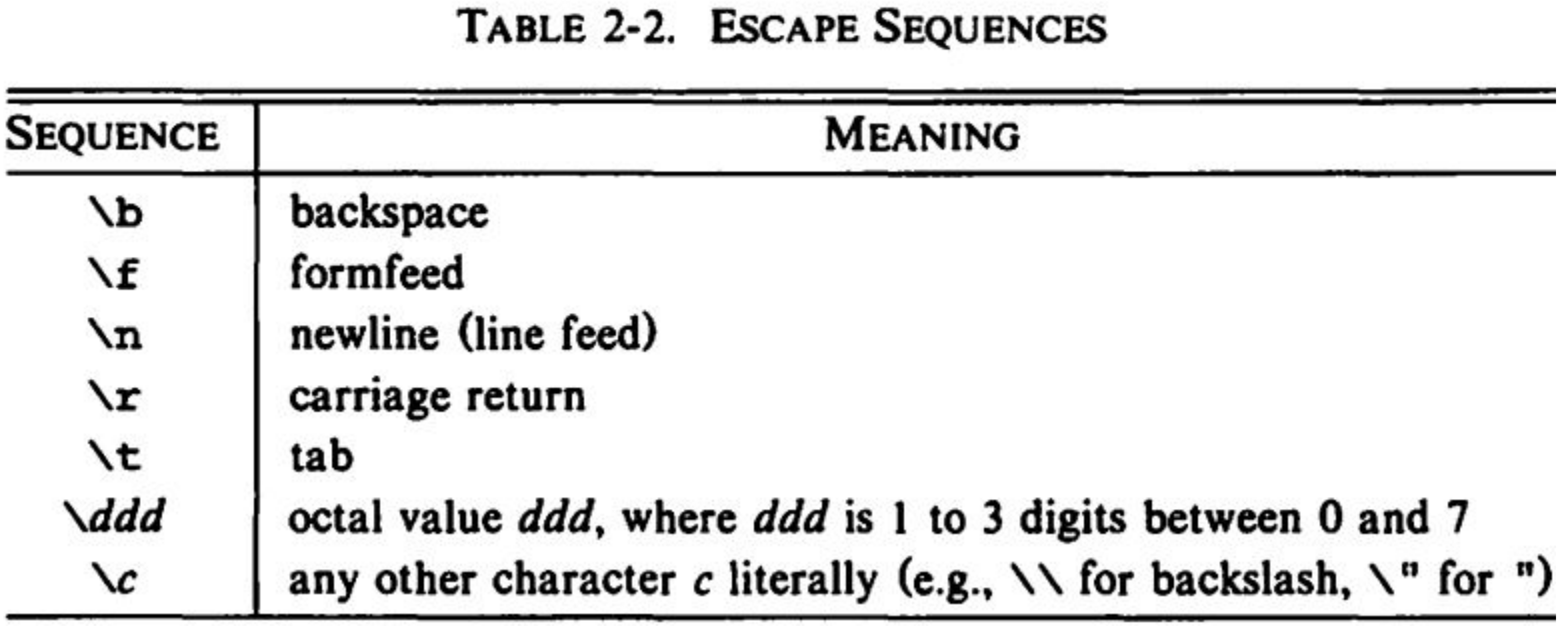


TABLE 2-2. ESCAPE SEQUENCES

SEQUENCE	MEANING
`\b`	backspace
`\f`	formfeed
`\n`	newline (line feed)
`\r`	carriage return
`\t`	tab
\ddd	octal value *ddd*, where *ddd* is 1 to 3 digits between 0 and 7
\c	any other character *c* literally (e.g., `\\` for backslash, `\"` for ")

Awk Range Patterns

- A range pattern consists of two patterns separated by a comma
- A range pattern matches each line between an occurrence of pattern 1 and the next occurrence of pattern 2 inclusive
- If no instance of the second pattern is subsequently found, then all lines to the end of the input are matched

Awk Patterns Summary

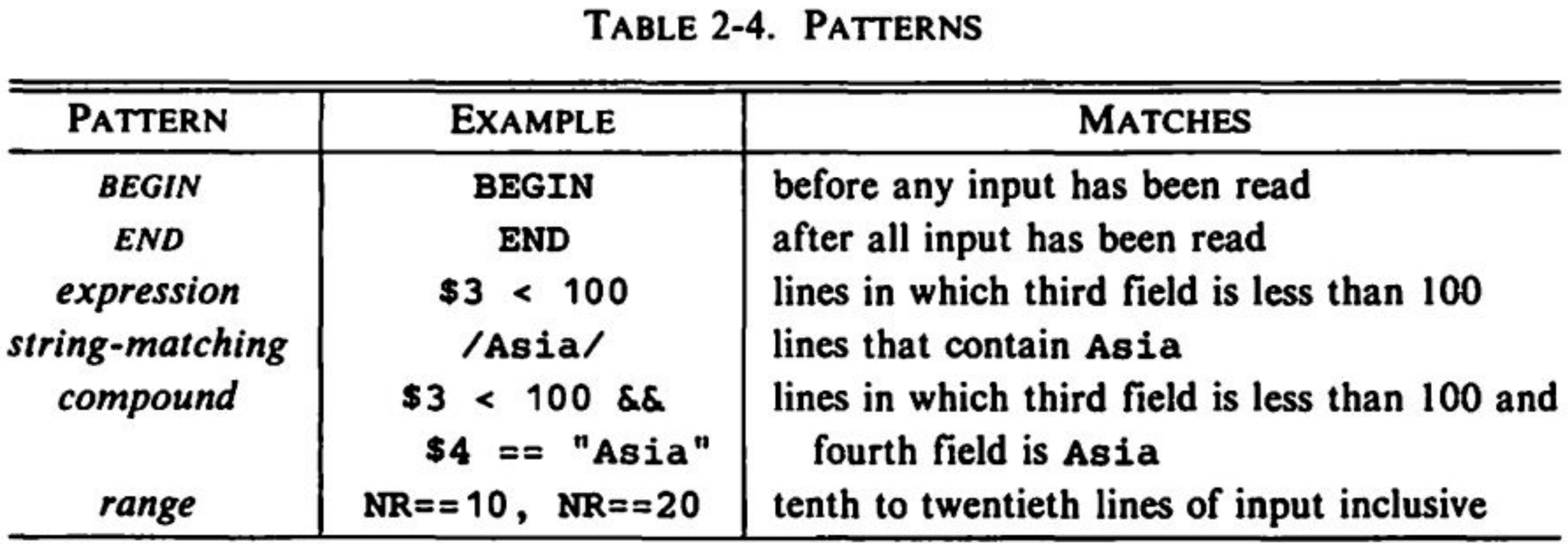


TABLE 2-4. PATTERNS

PATTERN	EXAMPLE	MATCHES
BEGIN	`BEGIN`	before any input has been read
END	`END`	after all input has been read
expression	`$3 < 100`	lines in which third field is less than 100
string-matching	`/Asia/`	lines that contain `Asia`
compound	`$3 < 100 && $4 == "Asia"`	lines in which third field is less than 100 and fourth field is `Asia`
range	`NR==10, NR==20`	tenth to twentieth lines of input inclusive

Awk Actions

- Executed if the pattern matches (if if there was no pattern)
- Are much like a typical language (such as C)
- Have access to a number of built in variables
- Can create variables or call functions (such as print)
- Parenthesis in function calls are optional
- Can override fields or create new fields

Actions

The statements in actions can include:

expressions, with constants, variables, assignments, function calls, etc.
`print` *expression-list*
`printf(`*format* `,` *expression-list* `)`
`if (`*expression*`)` *statement*
`if (`*expression*`)` *statement* `else` *statement*
`while (`*expression*`)` *statement*
`for (`*expression*`;` *expression*`;` *expression*`)` *statement*
`for (`*variable* `in` *array*`)` *statement*
`do` *statement* `while (`*expression*`)`
`break`
`continue`
`next`
`exit`
`exit` *expression*
`{` *statements* `}`

The simplest awk programs

- You've probably seen this before:
  - `awk ‘{ print $2 }’`

- Or maybe this:
  - `` awk `$3 == 10` ``

The simplest awk programs

- Print every line (not really helpful in the real world)
  - `awk ‘{ print }’`

- Equivalent to
  - `awk ‘{ print $0 }’`

The simplest awk programs

- Print some columns
  - `awk ‘{ print $1, $3 }’`
- Do some column math
  - `awk ‘{ print $1, $2 * $3 }’`

TABLE 2-5. BUILT-IN VARIABLES

VARIABLE	MEANING	DEFAULT
`ARGC`	number of command-line arguments	-
`ARGV`	array of command-line arguments	-
`FILENAME`	name of current input file	-
`FNR`	record number in current file	-
`FS`	controls the input field separator	`" "`
`NF`	number of fields in current record	-
`NR`	number of records read so far	-
`OFMT`	output format for numbers	`"%.6g"`
`OFS`	output field separator	`" "`
`ORS`	output record separator	`"\n"`
`RLENGTH`	length of string matched by `match` function	-
`RS`	controls the input record separator	`"\n"`
`RSTART`	start of string matched by `match` function	-
`SUBSEP`	subscript separator	`"\034"`

Magic variables!

- Print number of fields (columns)
    - `awk ‘{ print NF }’`

- Print number of lines read (basically line numbers)
    - `awk ‘{ print NR, $0 }’`

Add text to the output!

- Print number of fields (columns)
  - `awk '{ print $1 "makes" $3 "per hour" }'`

- More control with printf instead of print
  - `awk '{ printf("%s makes $%.2f per hour\n", $1, $3) }'`

Combine with other tools like sort and uniq

- Sort the output by $ per hour (3rd column)
  - `awk ‘{ print $1 “makes” $3 “per hour” } | sort -nk 3’`

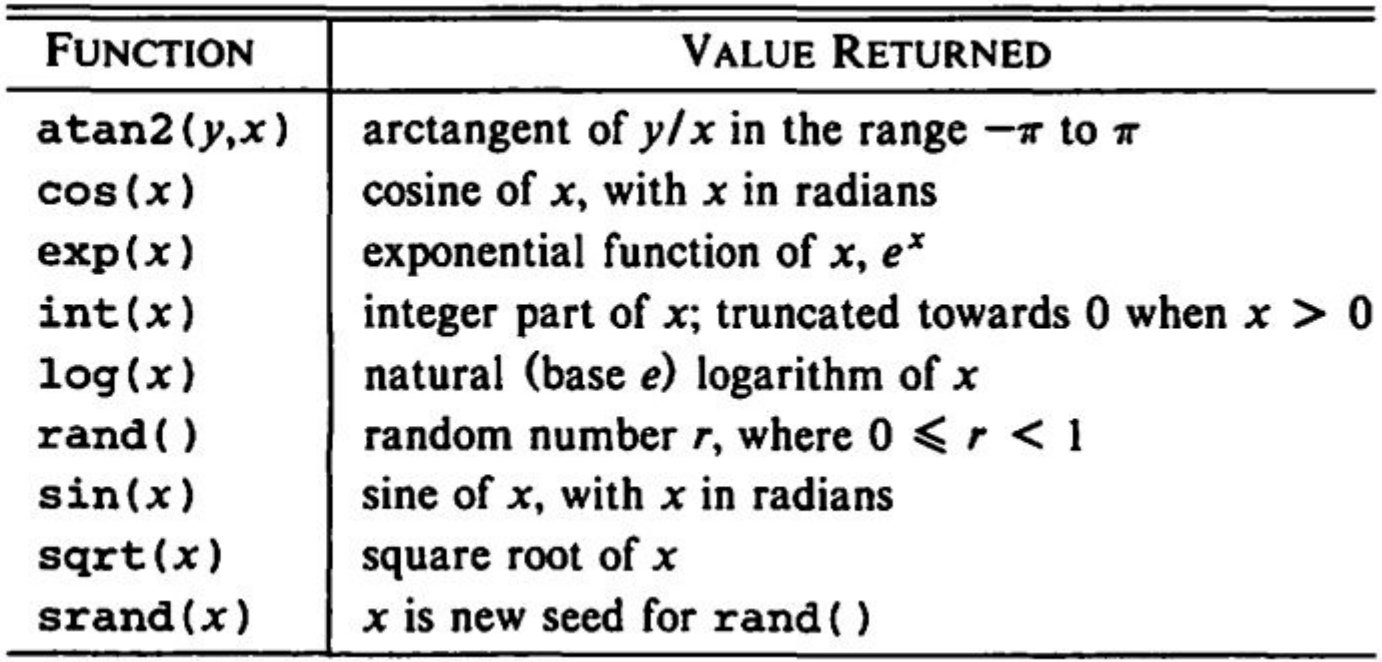
- Filter on unique wages
  - `awk ‘{ print $1 “makes” $3 “per hour” } | uniq -f 2’`

Expressions

1. The primary expressions are:
   numeric and string constants, variables, fields, function calls, array elements.

2. These operators combine expressions:
   assignment operators `=  +=  -=  *=  /=  %=  ^=`
   conditional expression operator `?:`
   logical operators `||` (OR), `&&` (AND), `!` (NOT)
   matching operators `~` and `!~`
   relational operators `<  <=  ==  !=  >  >=`
   concatenation (no explicit operator)
   arithmetic operators `+  -  *  /  %  ^`
   unary `+` and `-`
   increment and decrement operators `++` and `--` (prefix and postfix)
   parentheses for grouping

Built-in Math Functions



FUNCTION	VALUE RETURNED
`atan2(`y`,`x`)`	arctangent of y/x in the range $-\pi$ to π
`cos(`x`)`	cosine of x, with x in radians
`exp(`x`)`	exponential function of x, e^x
`int(`x`)`	integer part of x; truncated towards 0 when $x > 0$
`log(`x`)`	natural (base e) logarithm of x
`rand()`	random number r, where $0 \leqslant r < 1$
`sin(`x`)`	sine of x, with x in radians
`sqrt(`x`)`	square root of x
`srand(`x`)`	x is new seed for `rand()`

TABLE 2-7. BUILT-IN STRING FUNCTIONS

FUNCTION	DESCRIPTION
`gsub(`*r*`,`*s*`)`	substitute *s* for *r* globally in `$0`, return number of substitutions made
`gsub(`*r*`,`*s*`,`*t*`)`	substitute *s* for *r* globally in string *t*, return number of substitutions made
`index(`*s*`,`*t*`)`	return first position of string *t* in *s*, or 0 if *t* is not present
`length(`*s*`)`	return number of characters in *s*
`match(`*s*`,`*r*`)`	test whether *s* contains a substring matched by *r*; return index or 0; sets `RSTART` and `RLENGTH`
`split(`*s*`,`*a*`)`	split *s* into array *a* on `FS`, return number of fields
`split(`*s*`,`*a*`,`*fs*`)`	split *s* into array *a* on field separator *fs*, return number of fields
`sprintf(`*fmt*`,`*expr-list*`)`	return *expr-list* formatted according to format string *fmt*
`sub(`*r*`,`*s*`)`	substitute *s* for the leftmost longest substring of `$0` matched by *r*; return number of substitutions made
`sub(`*r*`,`*s*`,`*t*`)`	substitute *s* for the leftmost longest substring of *t* matched by *r*; return number of substitutions made
`substr(`*s*`,`*p*`)`	return suffix of *s* starting at position *p*
`substr(`*s*`,`*p*`,`*n*`)`	return substring of *s* of length *n* starting at position *p*

String functions

Implicit argument is $0 (the whole line):

```
    { gsub(/USA/, "United States"); print }
```

More examples:

```
    X = sprintf("%10s, %6d", $1, $2)

    gsub(/ana/, "anda", "banana")   # explicit argument
```

String Concatenation

Simply put two strings together:

Example: Concatenate fields 2 and 3:

```
    print $2 $3
```

Concatenate:

```
    print “hello” “world”
```

Outputs: `“helloworld”`

Types

Strings

```
    “String literal”
```

Numbers:

```
    +1  1. 0  1e0  0. 1e+ 1  10E-1  001
```

Types will be automatically coerced when needed.

TABLE 2-8. EXPRESSION OPERATORS

OPERATION	OPERATORS	EXAMPLE	MEANING OF EXAMPLE
assignment	= += -= *= /= %= ^=	x *= 2	x = x * 2
conditional	?:	x?y:z	if x is true then y else z
logical OR	¦¦	x ¦¦ y	1 if x or y is true, 0 otherwise
logical AND	&&	x && y	1 if x and y are true, 0 otherwise
array membership	in	i in a	1 if a[i] exists, 0 otherwise
matching	~ !~	$1 ~ /x/	1 if the first field contains an x, 0 otherwise
relational	< <= == != >= >	x == y	1 if x is equal to y, 0 otherwise
concatenation		"a" "bc"	"abc"; there is no explicit concatenation operator
add, subtract	+ -	x + y	sum of x and y
multiply, divide, mod	* / %	x % y	remainder of x divided by y
unary plus and minus	+ -	-x	negated value of x
logical NOT	!	!$1	1 if $1 is zero or null, 0 otherwise
exponentiation	^	x ^ y	x^y
increment, decrement	++ --	++x, x++	add 1 to x
field	$	$i+1	value of i-th field, plus 1
grouping	( )	($i)++	add 1 to value of i-th field

Control Flow

- Most standard control flow is supported
- Syntax is like C
- if/else
- while
- for

Control-Flow Statements

{ *statements* }
  statement grouping

`if` (*expression*) *statement*
  if *expression* is true, execute *statement*

`if` (*expression*) $statement_1$ `else` $statement_2$
  if *expression* is true, execute $statement_1$ otherwise execute $statement_2$

`while` (*expression*) *statement*
  if *expression* is true, execute *statement*, then repeat

`for` ($expression_1$; $expression_2$; $expression_3$) *statement*
  equivalent to $expression_1$; `while` ($expression_2$) { *statement*; $expression_3$ }

`for` (*variable* `in` *array*) *statement*
  execute *statement* with *variable* set to each subscript in *array* in turn

`do` *statement* `while` (*expression*)
  execute *statement*; if *expression* is true, repeat

`break`
  immediately leave innermost enclosing `while`, `for` or `do`

`continue`
  start next iteration of innermost enclosing `while`, `for` or `do`

`next`
  start next iteration of main input loop

`exit`

`exit` *expression*
  go immediately to the `END` action; if within the `END` action, exit program entirely. Return *expression* as program status.

Control Flow examples

`while` (*expression*)
    statement

`for` (*expression*$_1$; *expression*$_2$; *expression*$_3$)
    statement

```
{    i = 1
     while (i <= NF) {
          print $i
          i++
     }
}
```

```
{ for (i = 1; i <= NF; i++)
          print $i
}
```

Output Statements

`print`
    print `$0` on standard output

`print` *expression* `,` *expression* `,` ...
    print *expression*'s, separated by `OFS`, terminated by `ORS`

`print` *expression* `,` *expression* `,` ... `>`*filename*
    print on file *filename* instead of standard output

`print` *expression* `,` *expression* `,` ... `>>`*filename*
    append to file *filename* instead of overwriting previous contents

`print` *expression* `,` *expression* `,` ... `|` *command*
    print to standard input of *command*

`printf(`*format* `,` *expression* `,` *expression* `,` ... `)`
`printf(`*format* `,` *expression* `,` *expression* `,` ... `)` `>`*filename*
`printf(`*format* `,` *expression* `,` *expression* `,` ... `)` `>>`*filename*
`printf(`*format* `,` *expression* `,` *expression* `,` ... `)` `|` *command*
    `printf` statements are like `print` but the first argument specifies output format

`close(`*filename*`)`, `close(`*command*`)`
    break connection between `print` and *filename* or *command*

`system(`*command*`)`
    execute *command*; value is status return of *command*

Printf % characters

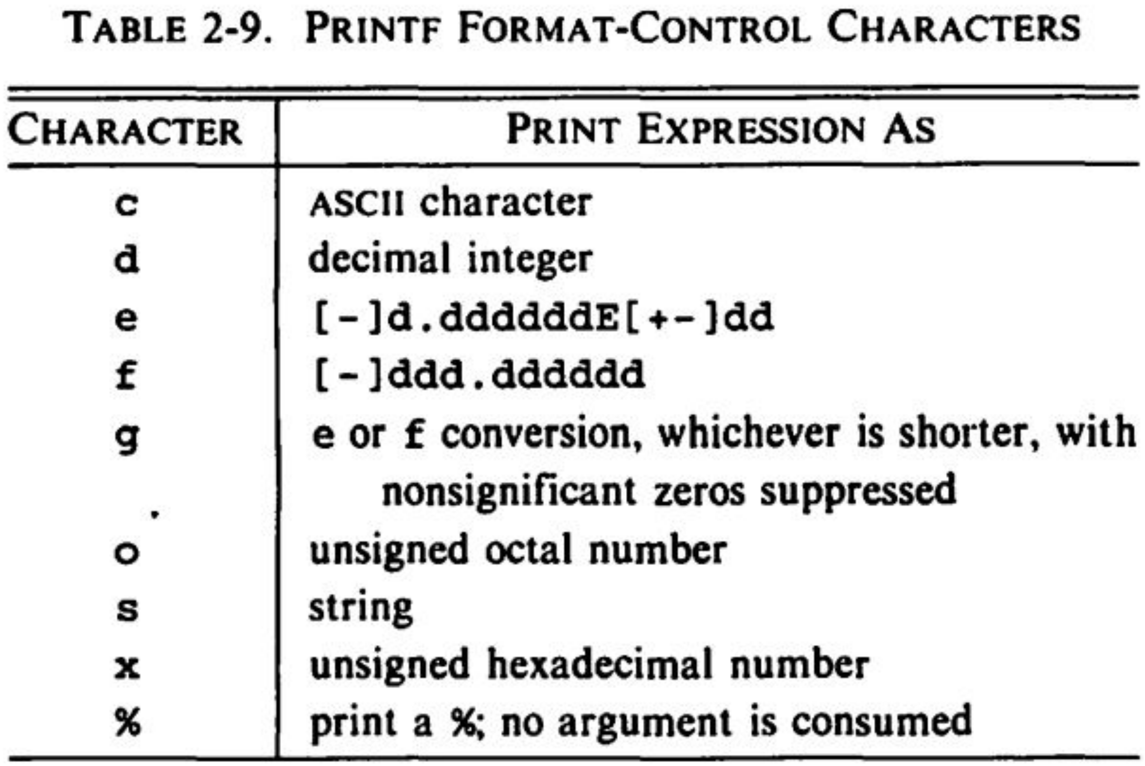


TABLE 2-9. PRINTF FORMAT-CONTROL CHARACTERS

CHARACTER	PRINT EXPRESSION AS
`c`	ASCII character
`d`	decimal integer
`e`	`[-]d.ddddddE[+-]dd`
`f`	`[-]ddd.dddddd`
`g`	`e` or `f` conversion, whichever is shorter, with nonsignificant zeros suppressed
`o`	unsigned octal number
`s`	string
`x`	unsigned hexadecimal number
`%`	print a %; no argument is consumed

Going Deeper

- We can write to files directly from awk:

```
(pattern) { print "expression" > "file name" }
```

- We can also pipe:

```
(pattern) { print "expression" | "command" }
```

Going Deeper - Variables

- We can also create and set variables:

```
{
    w += NF
    c = length + 1
}
```

We can call functions

- Count words in the input and print the number of lines, words, and characters (like wc):

```
{
    w += NF
    c += length + 1
}
END { print NR, w, c }
```

And Define Functions

- We can also define our own functions:

```
function add_three (number) {
    return number + 3
}

(pattern) { print add_three(36) }   # Outputs '''39'''
```

Going Deeper - Arrays

- Arrays are one dimensional
- For Strings or Numbers
- Arrays and elements do not need to be declared
- All arrays are associative
- Iterate with: `for (variable in array)`
- Delete element: `delete array[subscript]`
- `Array[“one”] = 2`
- `Array[5] = “two”`

Going Deeper - Field Manipulation

- Fields can be specified by expression:

  `$(NF-1)` is second to last, `$NF` is last, etc.

- A field variable referencing a non-existent field can be created through assignment. Initial value is empty string:

  ```
  $(NF+1) = $(NF-1) / 1000
  ```

Going Deeper - Self-contained Scripts

```
#!/usr/bin/awk -f

{ print $0 }
```

It can be invoked with: ./print.awk <filename>

The -f tells AWK that the argument that follows is the file to read the AWK program from, which is the same flag that is used in sed. Since they are often used for one-liners, both these programs default to executing a program given as a command-line argument, rather than a separate file.

Some weird Awk stuff

What the hell is this?

```
awk '{$1=$1}1' file.txt
```

It removes leading space. Easier to read as:

```
awk '{ $1=$1 }; { print }' file.txt
```

References

- *The AWK Programming Language* 1st Edition: Alfred V. Aho, Brian W. Kernighan, Peter J. Weinberger
- *Awk Tutorial* (2016): Jonathan Palardy - https://blog.jpalardy.com/posts/awk-tutorial-part-1/
- Awk (2019): Wikipedia - https://en.wikipedia.org/wiki/AWK

Challenges

Source: https://github.com/FreedomBen/awk-hack-the-planet

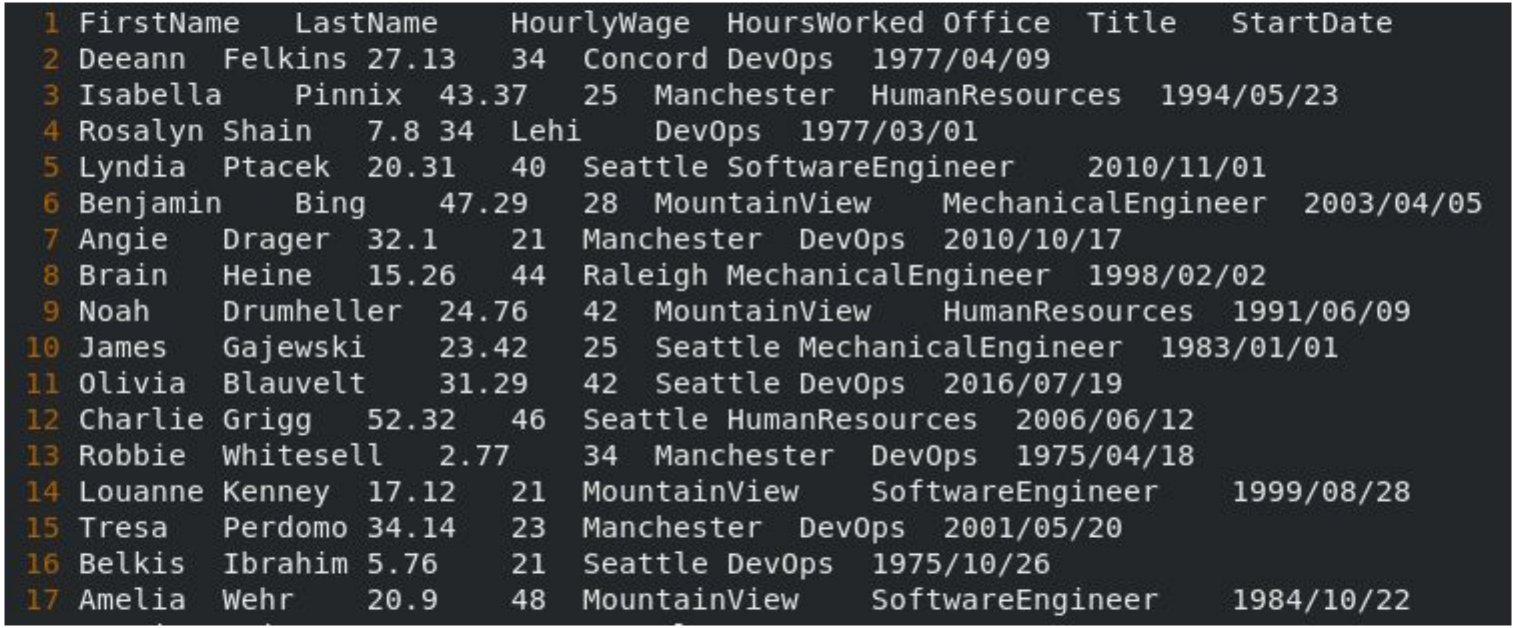
Scenario: Given a tab-separated value (tsv) file containing payroll information, provide some analysis. You should primarily use awk, but can combine with other tools (like sort, uniq). Don’t use grep or sed since awk can handle the same scenarios :-)

The file is payroll.tsv. You can generate a new one with the provided ruby script if you’d like to randomize it.

There are many different solutions. The ones presented are just mine. Many of them could be optimized and refactored to be more elegant.

Challenges - 01

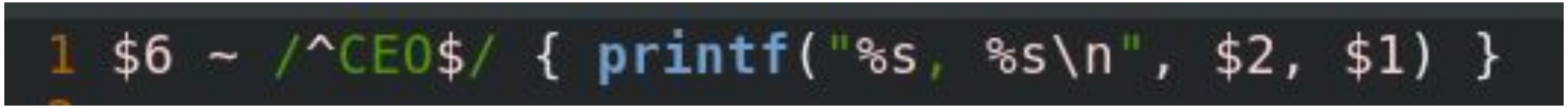
Q. What is the name of the CEO? Format like "LastName, FirstName"?



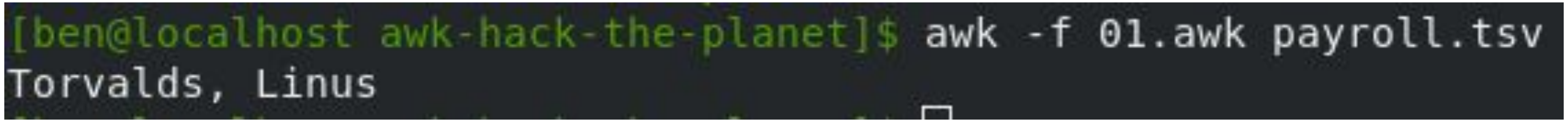
```
FirstName   LastName    HourlyWage  HoursWorked Office  Title   StartDate
Deeann  Felkins 27.13   34  Concord DevOps  1977/04/09
Isabella    Pinnix  43.37   25  Manchester  HumanResources  1994/05/23
Rosalyn Shain   7.8 34  Lehi    DevOps  1977/03/01
Lyndia  Ptacek  20.31   40  Seattle SoftwareEngineer    2010/11/01
Benjamin    Bing    47.29   28  MountainView    MechanicalEngineer  2003/04/05
Angie   Drager  32.1    21  Manchester  DevOps  2010/10/17
Brain   Heine   15.26   44  Raleigh MechanicalEngineer  1998/02/02
Noah    Drumheller  24.76   42  MountainView    HumanResources  1991/06/09
James   Gajewski    23.42   25  Seattle MechanicalEngineer  1983/01/01
Olivia  Blauvelt    31.29   42  Seattle DevOps  2016/07/19
Charlie Grigg   52.32   46  Seattle HumanResources  2006/06/12
Robbie  Whitesell   2.77    34  Manchester  DevOps  1975/04/18
Louanne Kenney  17.12   21  MountainView    SoftwareEngineer    1999/08/28
Tresa   Perdomo 34.14   23  Manchester  DevOps  2001/05/20
Belkis  Ibrahim 5.76    21  Seattle DevOps  1975/10/26
Amelia  Wehr    20.9    48  MountainView    SoftwareEngineer    1984/10/22
```

Challenges - 01

Q. What is the name of the CEO? Format like "LastName, FirstName"?



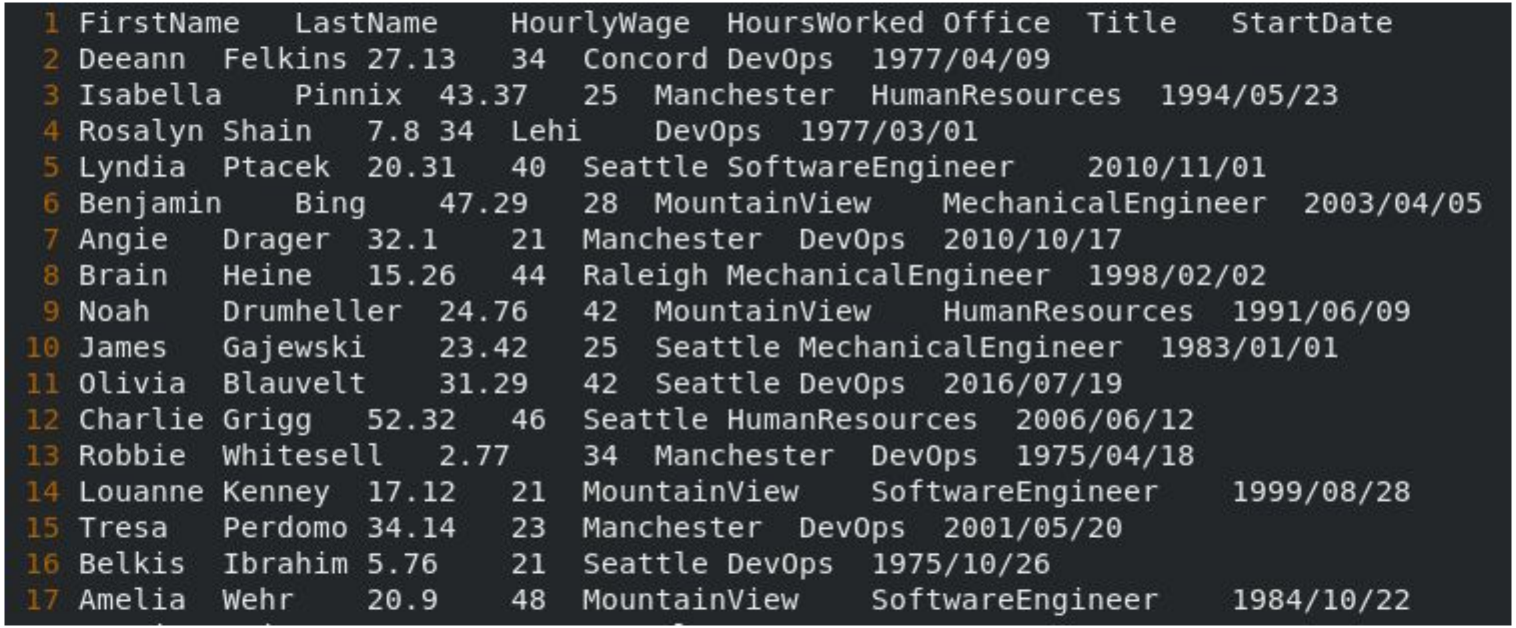
```
$6 ~ /^CEO$/ { printf("%s, %s\n", $2, $1) }
```



```
[ben@localhost awk-hack-the-planet]$ awk -f 01.awk payroll.tsv
Torvalds, Linus
```

Challenges - 02

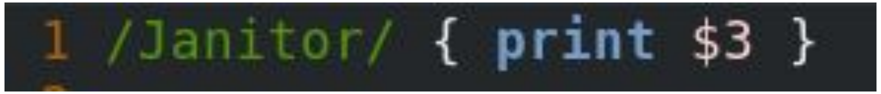
Q. How much money per hour does the janitor make?



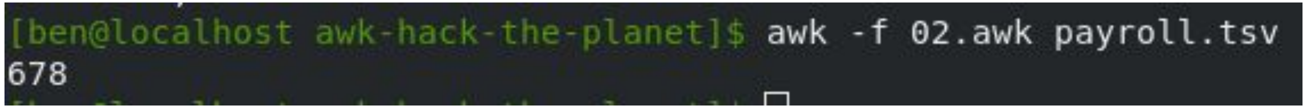
```
FirstName   LastName    HourlyWage  HoursWorked Office  Title   StartDate
Deeann  Felkins 27.13   34  Concord DevOps  1977/04/09
Isabella    Pinnix  43.37   25  Manchester  HumanResources  1994/05/23
Rosalyn Shain   7.8 34  Lehi    DevOps  1977/03/01
Lyndia  Ptacek  20.31   40  Seattle SoftwareEngineer    2010/11/01
Benjamin    Bing    47.29   28  MountainView    MechanicalEngineer  2003/04/05
Angie   Drager  32.1    21  Manchester  DevOps  2010/10/17
Brain   Heine   15.26   44  Raleigh MechanicalEngineer  1998/02/02
Noah    Drumheller  24.76   42  MountainView    HumanResources  1991/06/09
James   Gajewski    23.42   25  Seattle MechanicalEngineer  1983/01/01
Olivia  Blauvelt    31.29   42  Seattle DevOps  2016/07/19
Charlie Grigg   52.32   46  Seattle HumanResources  2006/06/12
Robbie  Whitesell   2.77    34  Manchester  DevOps  1975/04/18
Louanne Kenney  17.12   21  MountainView    SoftwareEngineer    1999/08/28
Tresa   Perdomo 34.14   23  Manchester  DevOps  2001/05/20
Belkis  Ibrahim 5.76    21  Seattle DevOps  1975/10/26
Amelia  Wehr    20.9    48  MountainView    SoftwareEngineer    1984/10/22
```

Challenges - 02

Q. How much money per hour does the janitor make?



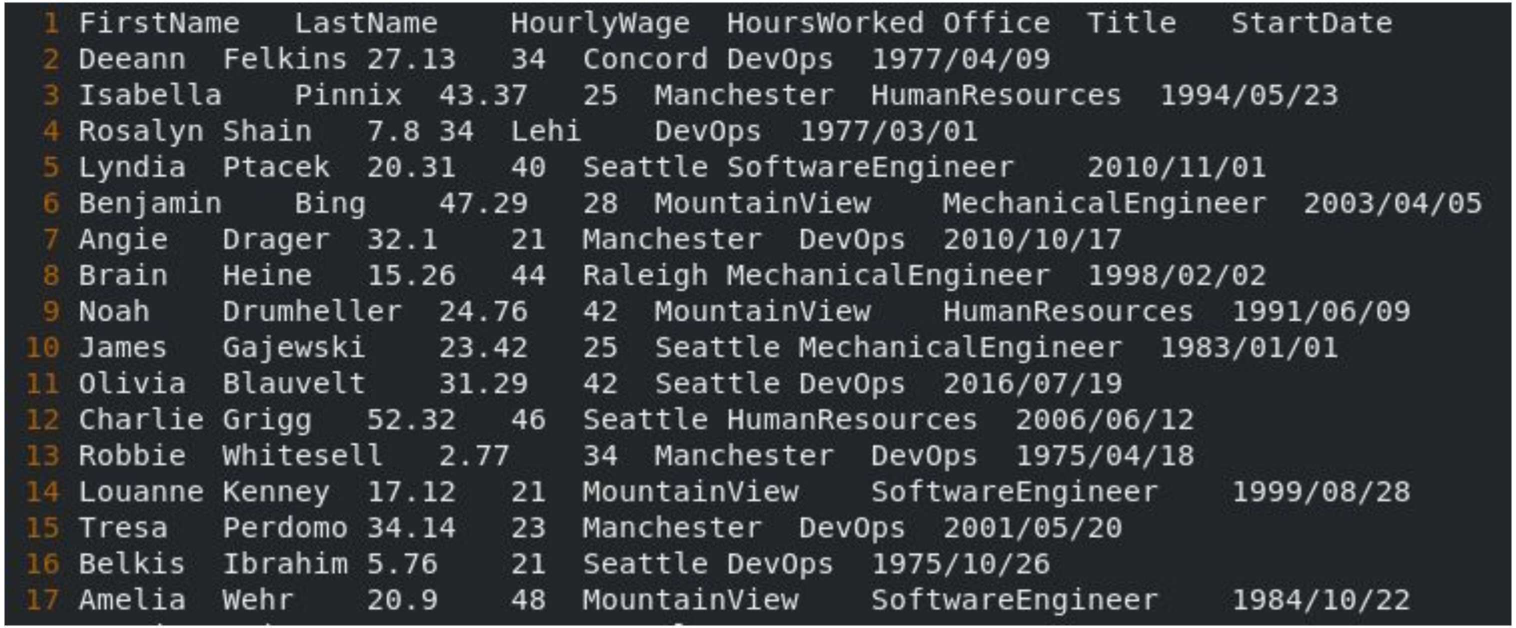
```
/Janitor/ { print $3 }
```



```
[ben@localhost awk-hack-the-planet]$ awk -f 02.awk payroll.tsv
678
```

Challenges - 03

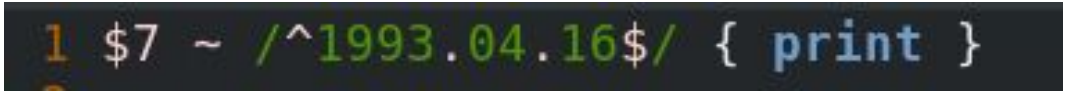
Q. Which employees were hired on April 16, 1993? (Print the list)



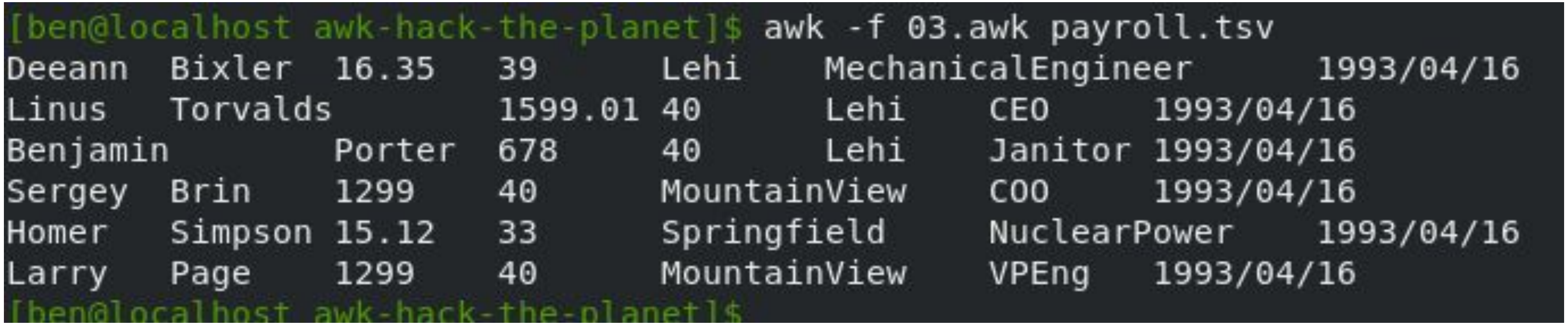
```
FirstName   LastName    HourlyWage  HoursWorked Office  Title   StartDate
Deeann  Felkins 27.13   34  Concord DevOps  1977/04/09
Isabella    Pinnix  43.37   25  Manchester  HumanResources  1994/05/23
Rosalyn Shain   7.8 34  Lehi    DevOps  1977/03/01
Lyndia  Ptacek  20.31   40  Seattle SoftwareEngineer    2010/11/01
Benjamin    Bing    47.29   28  MountainView    MechanicalEngineer  2003/04/05
Angie   Drager  32.1    21  Manchester  DevOps  2010/10/17
Brain   Heine   15.26   44  Raleigh MechanicalEngineer  1998/02/02
Noah    Drumheller  24.76   42  MountainView    HumanResources  1991/06/09
James   Gajewski    23.42   25  Seattle MechanicalEngineer  1983/01/01
Olivia  Blauvelt    31.29   42  Seattle DevOps  2016/07/19
Charlie Grigg   52.32   46  Seattle HumanResources  2006/06/12
Robbie  Whitesell   2.77    34  Manchester  DevOps  1975/04/18
Louanne Kenney  17.12   21  MountainView    SoftwareEngineer    1999/08/28
Tresa   Perdomo 34.14   23  Manchester  DevOps  2001/05/20
Belkis  Ibrahim 5.76    21  Seattle DevOps  1975/10/26
Amelia  Wehr    20.9    48  MountainView    SoftwareEngineer    1984/10/22
```

Challenges - 03

Q. Which employees were hired on April 16, 1993? (Print the list)



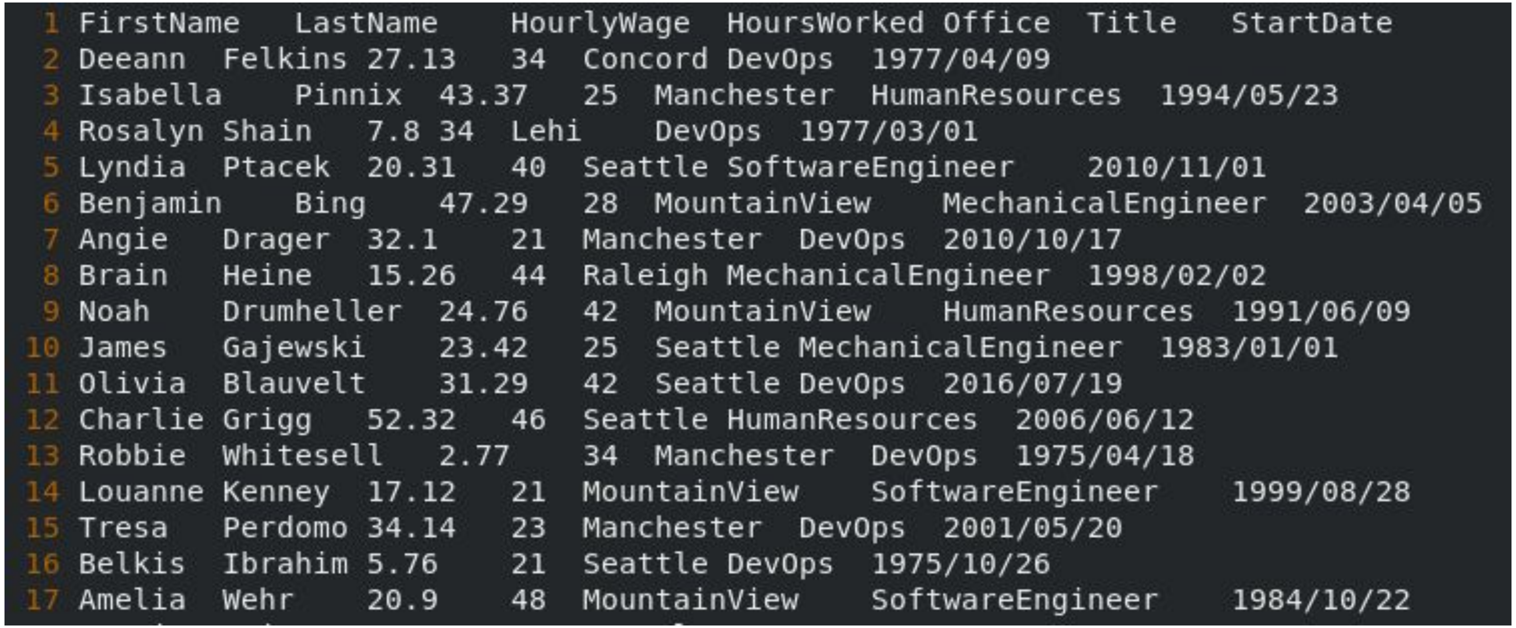
```
$7 ~ /^1993.04.16$/ { print }
```



```
[ben@localhost awk-hack-the-planet]$ awk -f 03.awk payroll.tsv
Deeann  Bixler  16.35   39      Lehi    MechanicalEngineer      1993/04/16
Linus   Torvalds        1599.01 40      Lehi    CEO     1993/04/16
Benjamin        Porter  678     40      Lehi    Janitor 1993/04/16
Sergey  Brin    1299    40      MountainView    COO     1993/04/16
Homer   Simpson 15.12   33      Springfield     NuclearPower    1993/04/16
Larry   Page    1299    40      MountainView    VPEng   1993/04/16
[ben@localhost awk-hack-the-planet]$
```

Challenges - 04

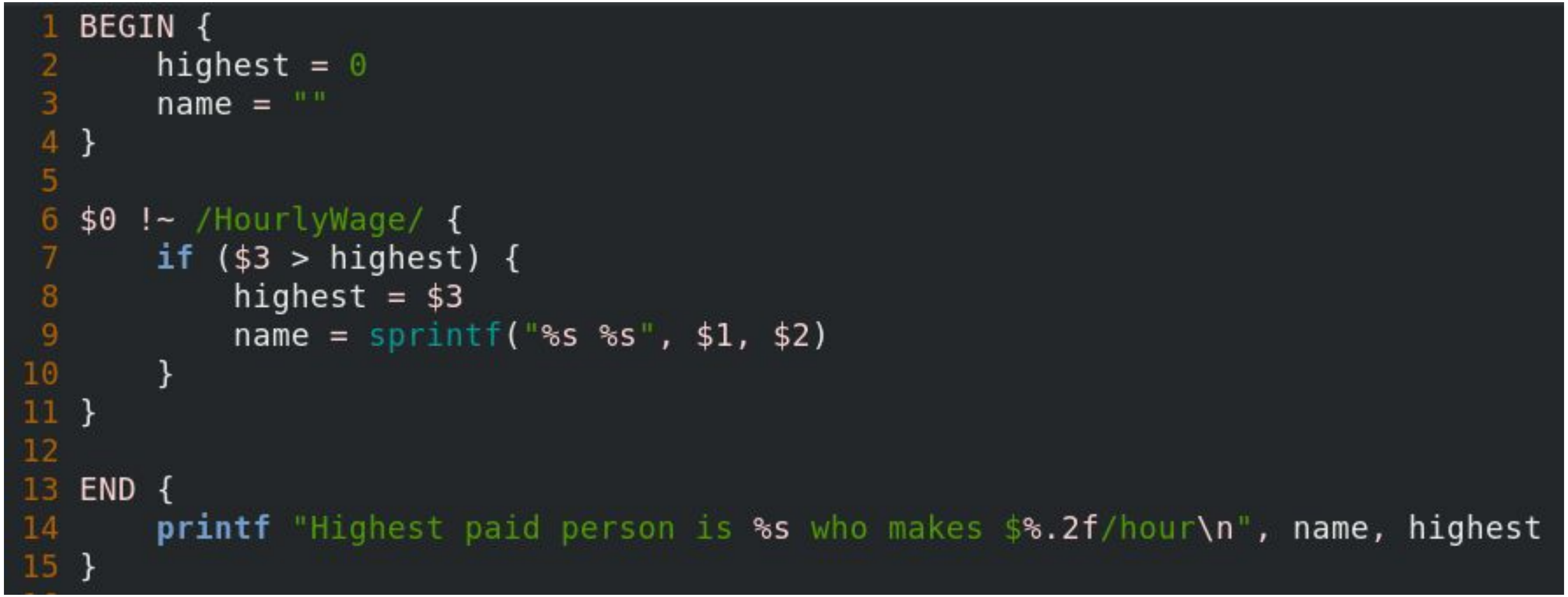
Q. Who is the highest paid employee?



```
FirstName   LastName    HourlyWage  HoursWorked Office  Title   StartDate
Deeann  Felkins 27.13   34  Concord DevOps  1977/04/09
Isabella    Pinnix  43.37   25  Manchester  HumanResources  1994/05/23
Rosalyn Shain   7.8 34  Lehi    DevOps  1977/03/01
Lyndia  Ptacek  20.31   40  Seattle SoftwareEngineer    2010/11/01
Benjamin    Bing    47.29   28  MountainView    MechanicalEngineer  2003/04/05
Angie   Drager  32.1    21  Manchester  DevOps  2010/10/17
Brain   Heine   15.26   44  Raleigh MechanicalEngineer  1998/02/02
Noah    Drumheller  24.76   42  MountainView    HumanResources  1991/06/09
James   Gajewski    23.42   25  Seattle MechanicalEngineer  1983/01/01
Olivia  Blauvelt    31.29   42  Seattle DevOps  2016/07/19
Charlie Grigg   52.32   46  Seattle HumanResources  2006/06/12
Robbie  Whitesell   2.77    34  Manchester  DevOps  1975/04/18
Louanne Kenney  17.12   21  MountainView    SoftwareEngineer    1999/08/28
Tresa   Perdomo 34.14   23  Manchester  DevOps  2001/05/20
Belkis  Ibrahim 5.76    21  Seattle DevOps  1975/10/26
Amelia  Wehr    20.9    48  MountainView    SoftwareEngineer    1984/10/22
```

Challenges - 04

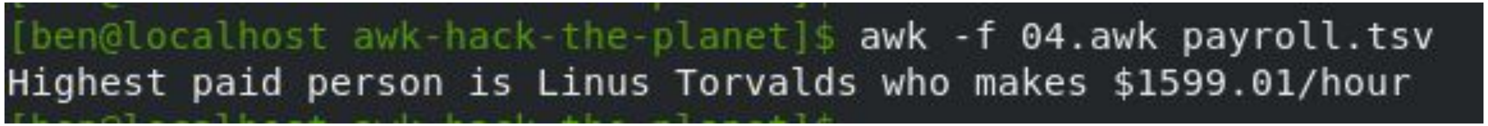
Q. Who is the highest paid employee?



```
BEGIN {
    highest = 0
    name = ""
}

$0 !~ /HourlyWage/ {
    if ($3 > highest) {
        highest = $3
        name = sprintf("%s %s", $1, $2)
    }
}

END {
    printf "Highest paid person is %s who makes $%.2f/hour\n", name, highest
}
```

Challenges - 04

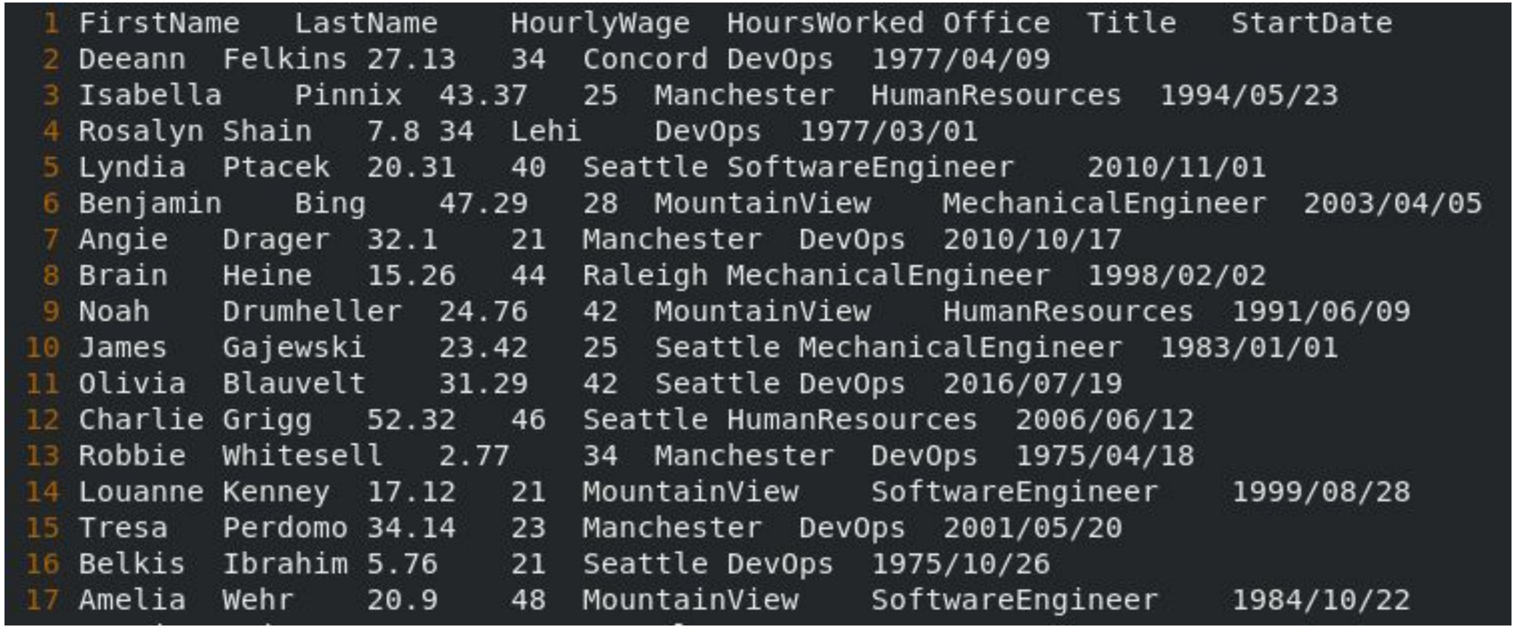
Q. Who is the highest paid employee?



```
[ben@localhost awk-hack-the-planet]$ awk -f 04.awk payroll.tsv
Highest paid person is Linus Torvalds who makes $1599.01/hour
```

Challenges - 05

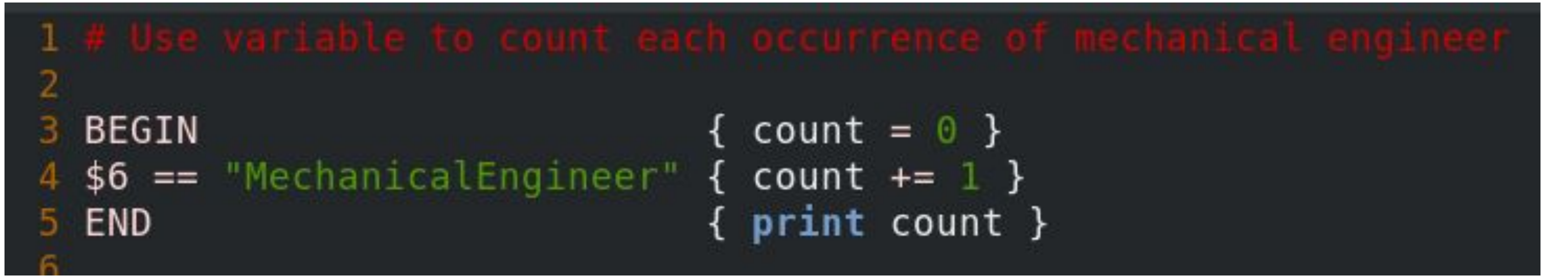
Q. How many mechanical engineers work here?



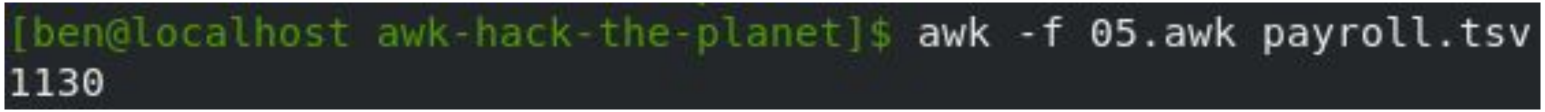
```
FirstName	LastName	HourlyWage	HoursWorked	Office	Title	StartDate
Deeann	Felkins	27.13	34	Concord	DevOps	1977/04/09
Isabella	Pinnix	43.37	25	Manchester	HumanResources	1994/05/23
Rosalyn	Shain	7.8	34	Lehi	DevOps	1977/03/01
Lyndia	Ptacek	20.31	40	Seattle	SoftwareEngineer	2010/11/01
Benjamin	Bing	47.29	28	MountainView	MechanicalEngineer	2003/04/05
Angie	Drager	32.1	21	Manchester	DevOps	2010/10/17
Brain	Heine	15.26	44	Raleigh	MechanicalEngineer	1998/02/02
Noah	Drumheller	24.76	42	MountainView	HumanResources	1991/06/09
James	Gajewski	23.42	25	Seattle	MechanicalEngineer	1983/01/01
Olivia	Blauvelt	31.29	42	Seattle	DevOps	2016/07/19
Charlie	Grigg	52.32	46	Seattle	HumanResources	2006/06/12
Robbie	Whitesell	2.77	34	Manchester	DevOps	1975/04/18
Louanne	Kenney	17.12	21	MountainView	SoftwareEngineer	1999/08/28
Tresa	Perdomo	34.14	23	Manchester	DevOps	2001/05/20
Belkis	Ibrahim	5.76	21	Seattle	DevOps	1975/10/26
Amelia	Wehr	20.9	48	MountainView	SoftwareEngineer	1984/10/22
```

Challenges - 05

Q. How many mechanical engineers work here?



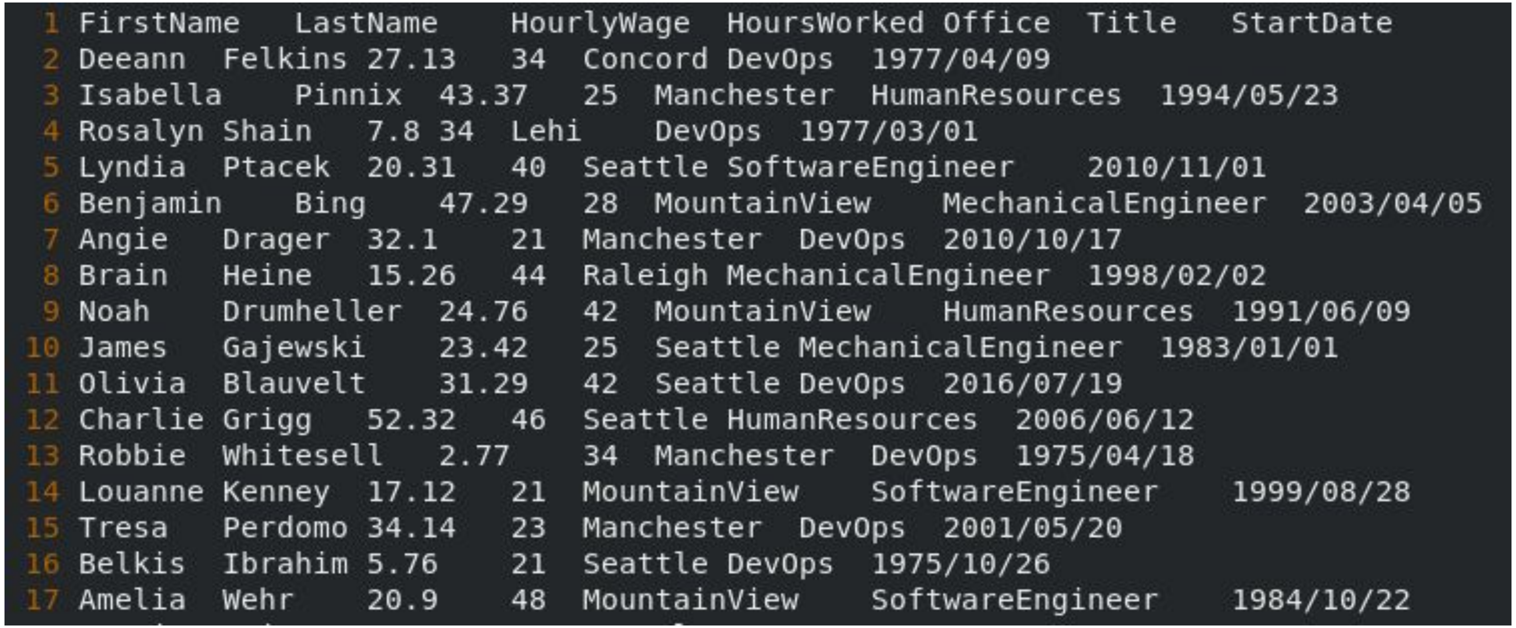
```
# Use variable to count each occurrence of mechanical engineer

BEGIN                          { count = 0 }
$6 == "MechanicalEngineer"     { count += 1 }
END                            { print count }
```



```
[ben@localhost awk-hack-the-planet]$ awk -f 05.awk payroll.tsv
1130
```

Challenges - 06

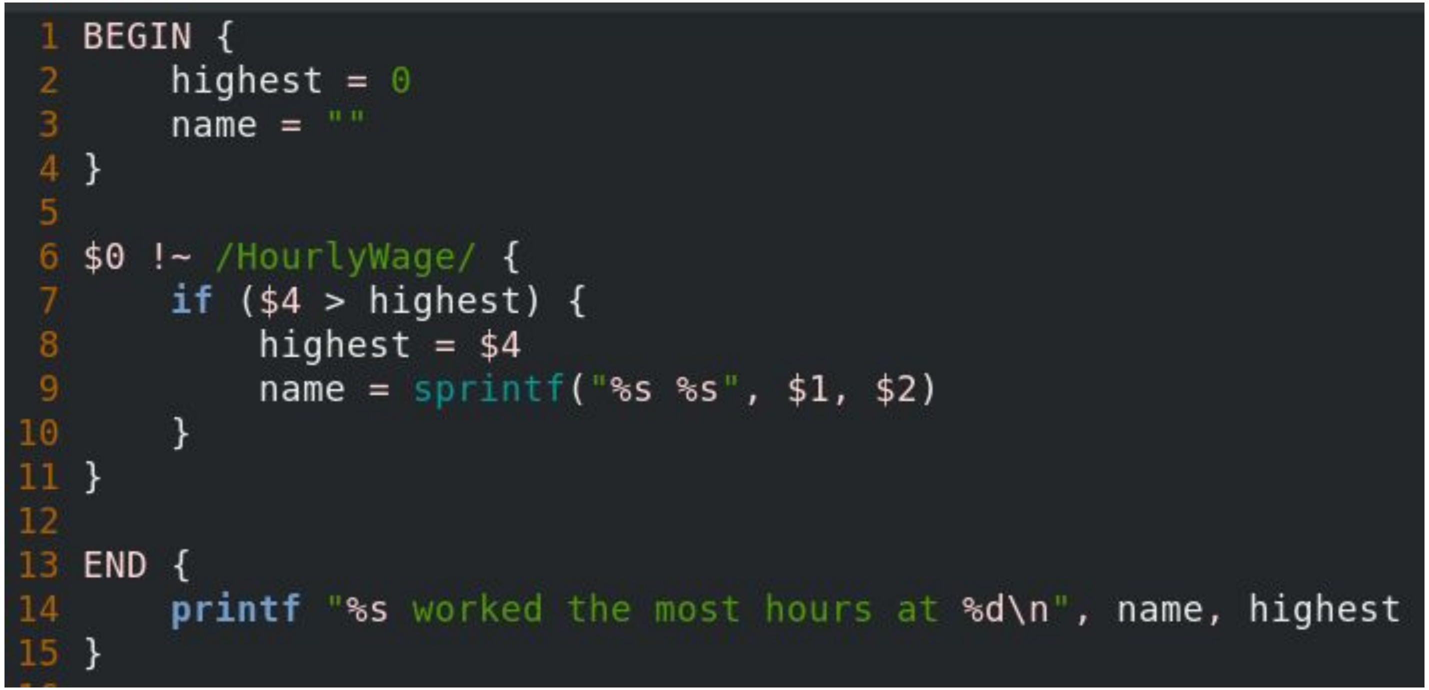
Q. Who worked the most hours this week?



```
FirstName   LastName    HourlyWage  HoursWorked Office  Title   StartDate
Deeann  Felkins 27.13   34  Concord DevOps  1977/04/09
Isabella    Pinnix  43.37   25  Manchester  HumanResources  1994/05/23
Rosalyn Shain   7.8 34  Lehi    DevOps  1977/03/01
Lyndia  Ptacek  20.31   40  Seattle SoftwareEngineer    2010/11/01
Benjamin    Bing    47.29   28  MountainView    MechanicalEngineer  2003/04/05
Angie   Drager  32.1    21  Manchester  DevOps  2010/10/17
Brain   Heine   15.26   44  Raleigh MechanicalEngineer  1998/02/02
Noah    Drumheller  24.76   42  MountainView    HumanResources  1991/06/09
James   Gajewski    23.42   25  Seattle MechanicalEngineer  1983/01/01
Olivia  Blauvelt    31.29   42  Seattle DevOps  2016/07/19
Charlie Grigg   52.32   46  Seattle HumanResources  2006/06/12
Robbie  Whitesell   2.77    34  Manchester  DevOps  1975/04/18
Louanne Kenney  17.12   21  MountainView    SoftwareEngineer    1999/08/28
Tresa   Perdomo 34.14   23  Manchester  DevOps  2001/05/20
Belkis  Ibrahim 5.76    21  Seattle DevOps  1975/10/26
Amelia  Wehr    20.9    48  MountainView    SoftwareEngineer    1984/10/22
```

Challenges - 06

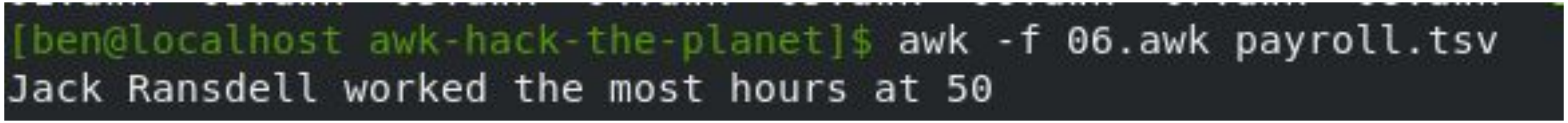
Q. Who worked the most hours this week?



```
BEGIN {
    highest = 0
    name = ""
}

$0 !~ /HourlyWage/ {
    if ($4 > highest) {
        highest = $4
        name = sprintf("%s %s", $1, $2)
    }
}

END {
    printf "%s worked the most hours at %d\n", name, highest
}
```

Challenges - 06

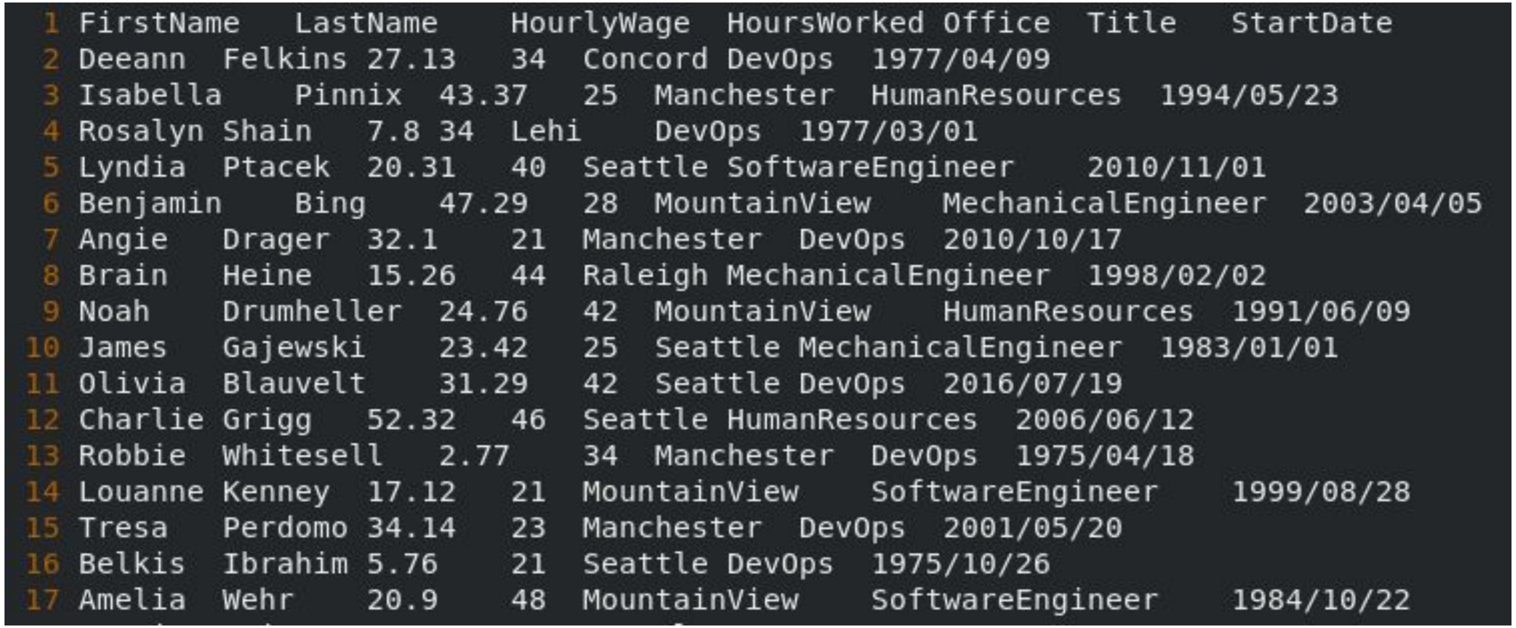
Q. Who worked the most hours this week?



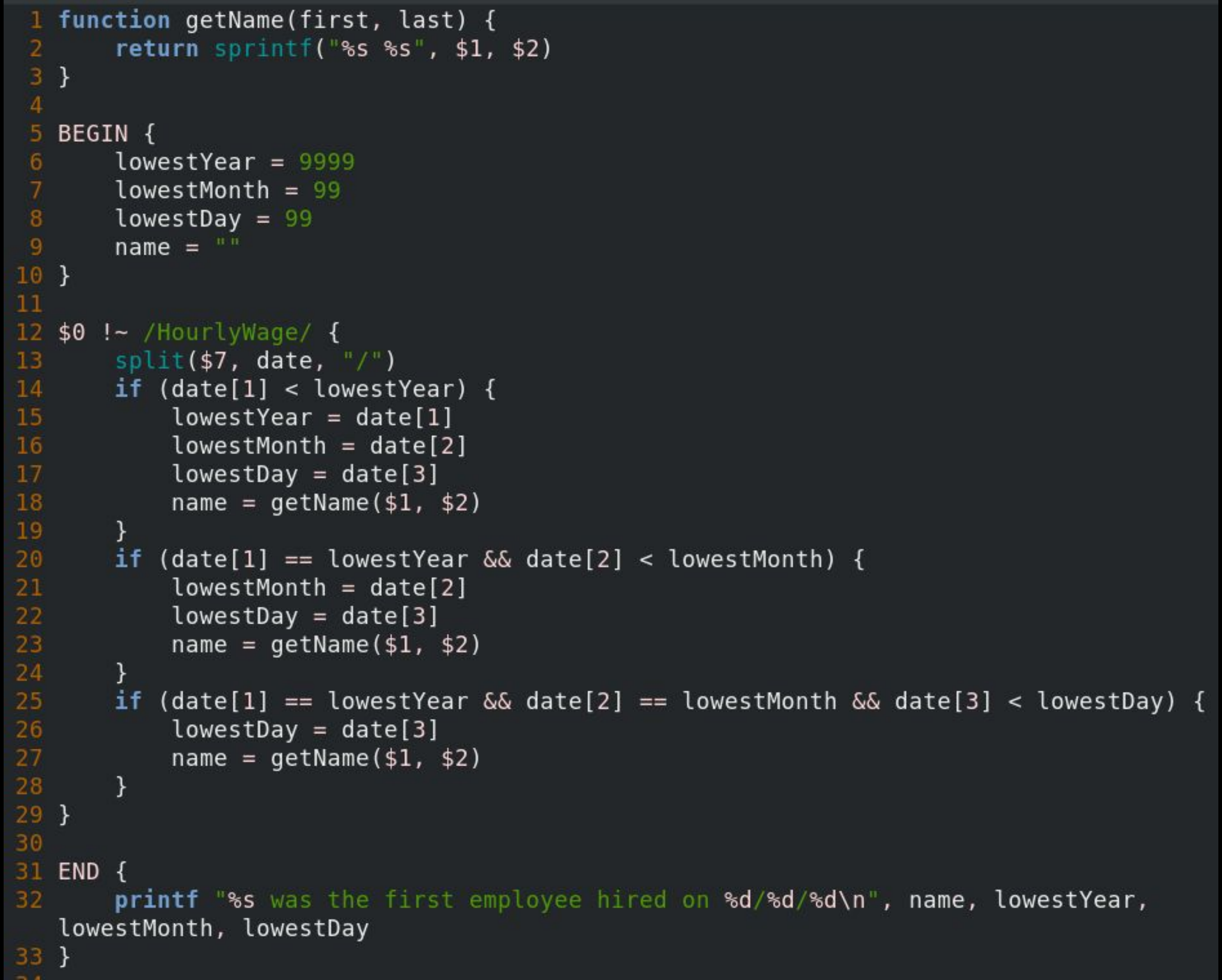
```
[ben@localhost awk-hack-the-planet]$ awk -f 06.awk payroll.tsv
Jack Ransdell worked the most hours at 50
```

Challenges - 07

Q. Who was the first employee hired?



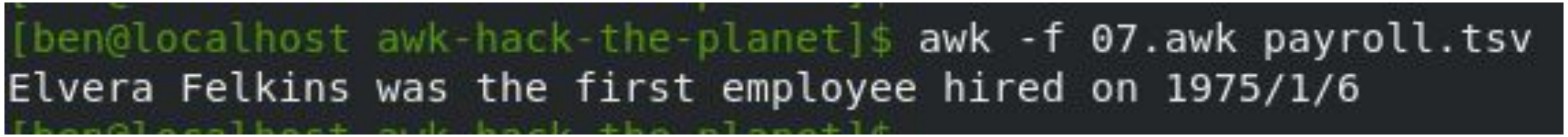
```
FirstName   LastName    HourlyWage  HoursWorked Office  Title   StartDate
Deeann  Felkins 27.13   34  Concord DevOps  1977/04/09
Isabella    Pinnix  43.37   25  Manchester  HumanResources  1994/05/23
Rosalyn Shain   7.8 34  Lehi    DevOps  1977/03/01
Lyndia  Ptacek  20.31   40  Seattle SoftwareEngineer    2010/11/01
Benjamin    Bing    47.29   28  MountainView    MechanicalEngineer  2003/04/05
Angie   Drager  32.1    21  Manchester  DevOps  2010/10/17
Brain   Heine   15.26   44  Raleigh MechanicalEngineer  1998/02/02
Noah    Drumheller  24.76   42  MountainView    HumanResources  1991/06/09
James   Gajewski    23.42   25  Seattle MechanicalEngineer  1983/01/01
Olivia  Blauvelt    31.29   42  Seattle DevOps  2016/07/19
Charlie Grigg   52.32   46  Seattle HumanResources  2006/06/12
Robbie  Whitesell   2.77    34  Manchester  DevOps  1975/04/18
Louanne Kenney  17.12   21  MountainView    SoftwareEngineer    1999/08/28
Tresa   Perdomo 34.14   23  Manchester  DevOps  2001/05/20
Belkis  Ibrahim 5.76    21  Seattle DevOps  1975/10/26
Amelia  Wehr    20.9    48  MountainView    SoftwareEngineer    1984/10/22
```



```
function getName(first, last) {
    return sprintf("%s %s", $1, $2)
}

BEGIN {
    lowestYear = 9999
    lowestMonth = 99
    lowestDay = 99
    name = ""
}

$0 !~ /HourlyWage/ {
    split($7, date, "/")
    if (date[1] < lowestYear) {
        lowestYear = date[1]
        lowestMonth = date[2]
        lowestDay = date[3]
        name = getName($1, $2)
    }
    if (date[1] == lowestYear && date[2] < lowestMonth) {
        lowestMonth = date[2]
        lowestDay = date[3]
        name = getName($1, $2)
    }
    if (date[1] == lowestYear && date[2] == lowestMonth && date[3] < lowestDay) {
        lowestDay = date[3]
        name = getName($1, $2)
    }
}

END {
    printf "%s was the first employee hired on %d/%d/%d\n", name, lowestYear, lowestMonth, lowestDay
}
```

Challenges - 07

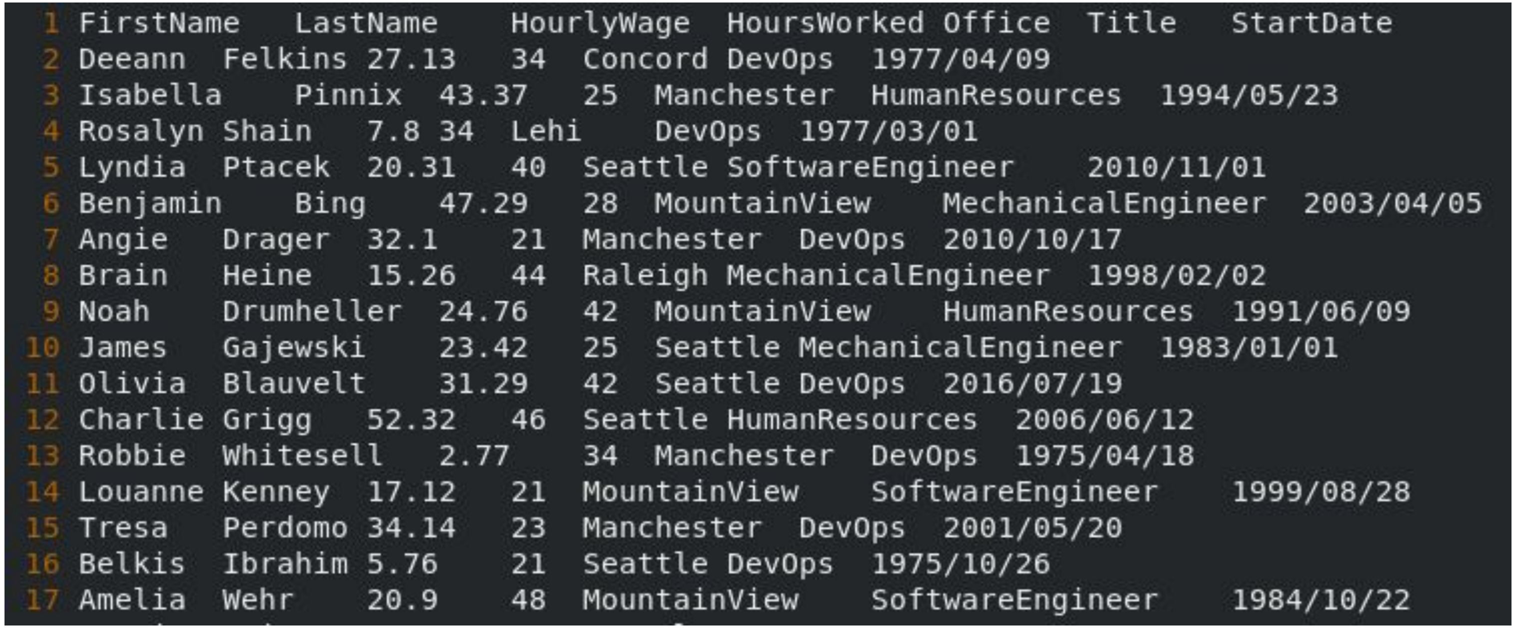
Q. Who was the first employee hired?



```
[ben@localhost awk-hack-the-planet]$ awk -f 07.awk payroll.tsv
Elvera Felkins was the first employee hired on 1975/1/6
```

Challenges - 08

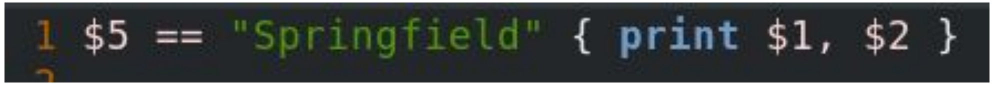
Q. Which employee works in the Springfield office?



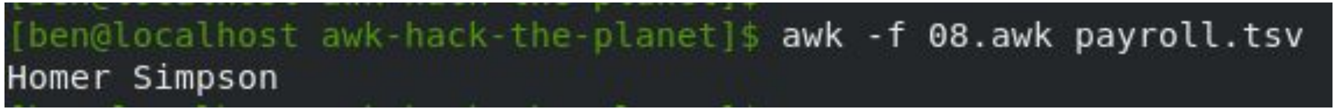
```
FirstName   LastName    HourlyWage  HoursWorked Office  Title   StartDate
Deeann  Felkins 27.13   34  Concord DevOps  1977/04/09
Isabella    Pinnix  43.37   25  Manchester  HumanResources  1994/05/23
Rosalyn Shain   7.8 34  Lehi    DevOps  1977/03/01
Lyndia  Ptacek  20.31   40  Seattle SoftwareEngineer    2010/11/01
Benjamin    Bing    47.29   28  MountainView    MechanicalEngineer  2003/04/05
Angie   Drager  32.1    21  Manchester  DevOps  2010/10/17
Brain   Heine   15.26   44  Raleigh MechanicalEngineer  1998/02/02
Noah    Drumheller  24.76   42  MountainView    HumanResources  1991/06/09
James   Gajewski    23.42   25  Seattle MechanicalEngineer  1983/01/01
Olivia  Blauvelt    31.29   42  Seattle DevOps  2016/07/19
Charlie Grigg   52.32   46  Seattle HumanResources  2006/06/12
Robbie  Whitesell   2.77    34  Manchester  DevOps  1975/04/18
Louanne Kenney  17.12   21  MountainView    SoftwareEngineer    1999/08/28
Tresa   Perdomo 34.14   23  Manchester  DevOps  2001/05/20
Belkis  Ibrahim 5.76    21  Seattle DevOps  1975/10/26
Amelia  Wehr    20.9    48  MountainView    SoftwareEngineer    1984/10/22
```

Challenges - 08

Q. Which employee works in the Springfield office?



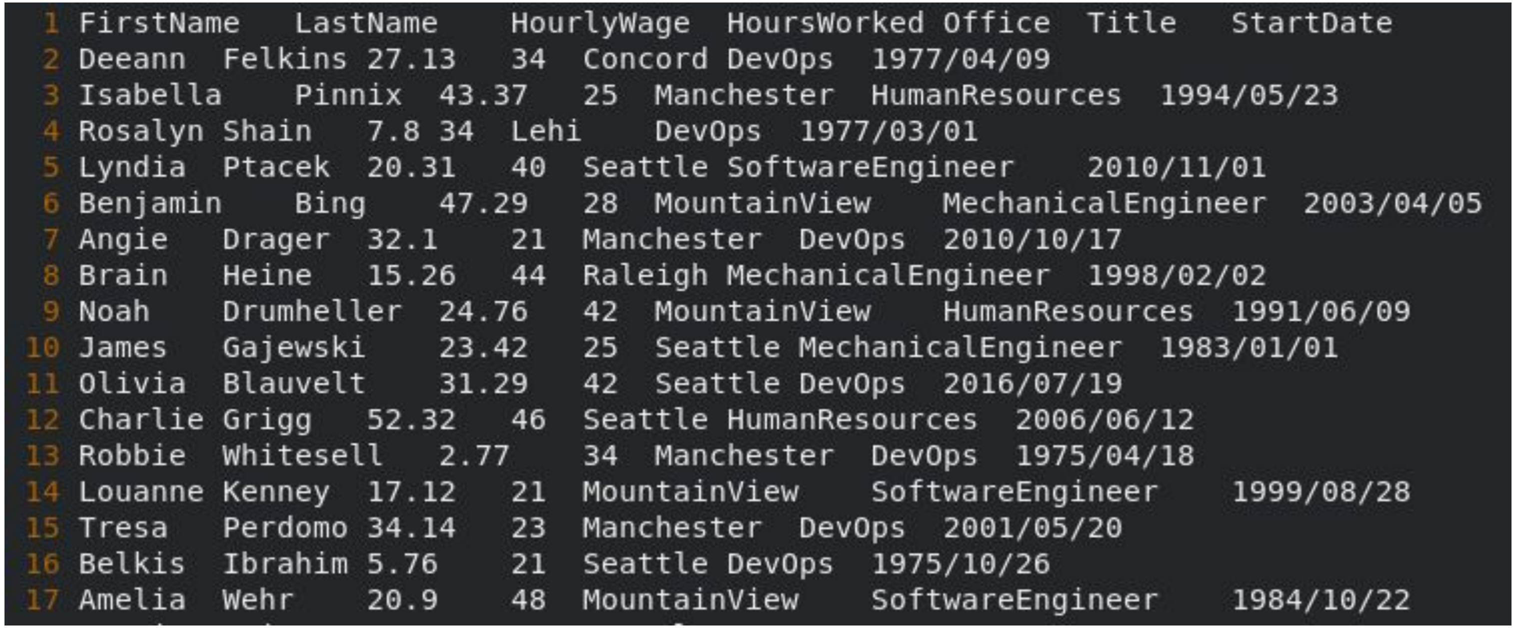
```
$5 == "Springfield" { print $1, $2 }
```



```
[ben@localhost awk-hack-the-planet]$ awk -f 08.awk payroll.tsv
Homer Simpson
```

Challenges - 09

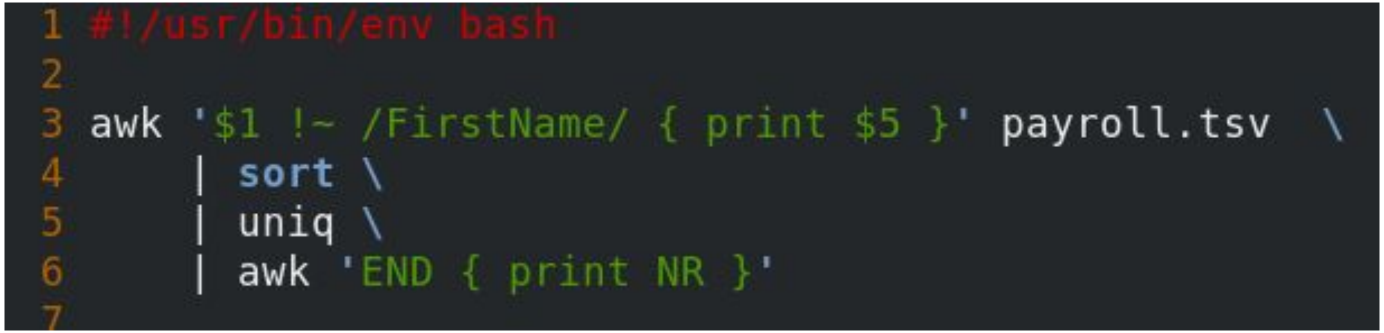
Q. How many different office locations does the company have?



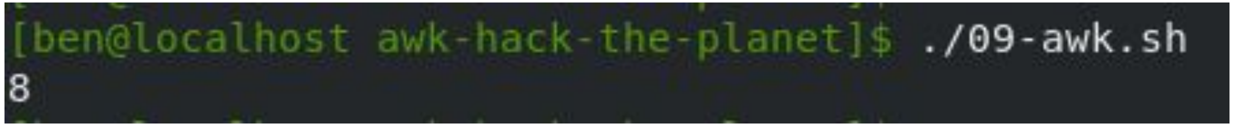
```
FirstName	LastName	HourlyWage	HoursWorked	Office	Title	StartDate
Deeann	Felkins	27.13	34	Concord	DevOps	1977/04/09
Isabella	Pinnix	43.37	25	Manchester	HumanResources	1994/05/23
Rosalyn	Shain	7.8	34	Lehi	DevOps	1977/03/01
Lyndia	Ptacek	20.31	40	Seattle	SoftwareEngineer	2010/11/01
Benjamin	Bing	47.29	28	MountainView	MechanicalEngineer	2003/04/05
Angie	Drager	32.1	21	Manchester	DevOps	2010/10/17
Brain	Heine	15.26	44	Raleigh	MechanicalEngineer	1998/02/02
Noah	Drumheller	24.76	42	MountainView	HumanResources	1991/06/09
James	Gajewski	23.42	25	Seattle	MechanicalEngineer	1983/01/01
Olivia	Blauvelt	31.29	42	Seattle	DevOps	2016/07/19
Charlie	Grigg	52.32	46	Seattle	HumanResources	2006/06/12
Robbie	Whitesell	2.77	34	Manchester	DevOps	1975/04/18
Louanne	Kenney	17.12	21	MountainView	SoftwareEngineer	1999/08/28
Tresa	Perdomo	34.14	23	Manchester	DevOps	2001/05/20
Belkis	Ibrahim	5.76	21	Seattle	DevOps	1975/10/26
Amelia	Wehr	20.9	48	MountainView	SoftwareEngineer	1984/10/22
```

Challenges - 09

Q. How many different office locations does the company have?



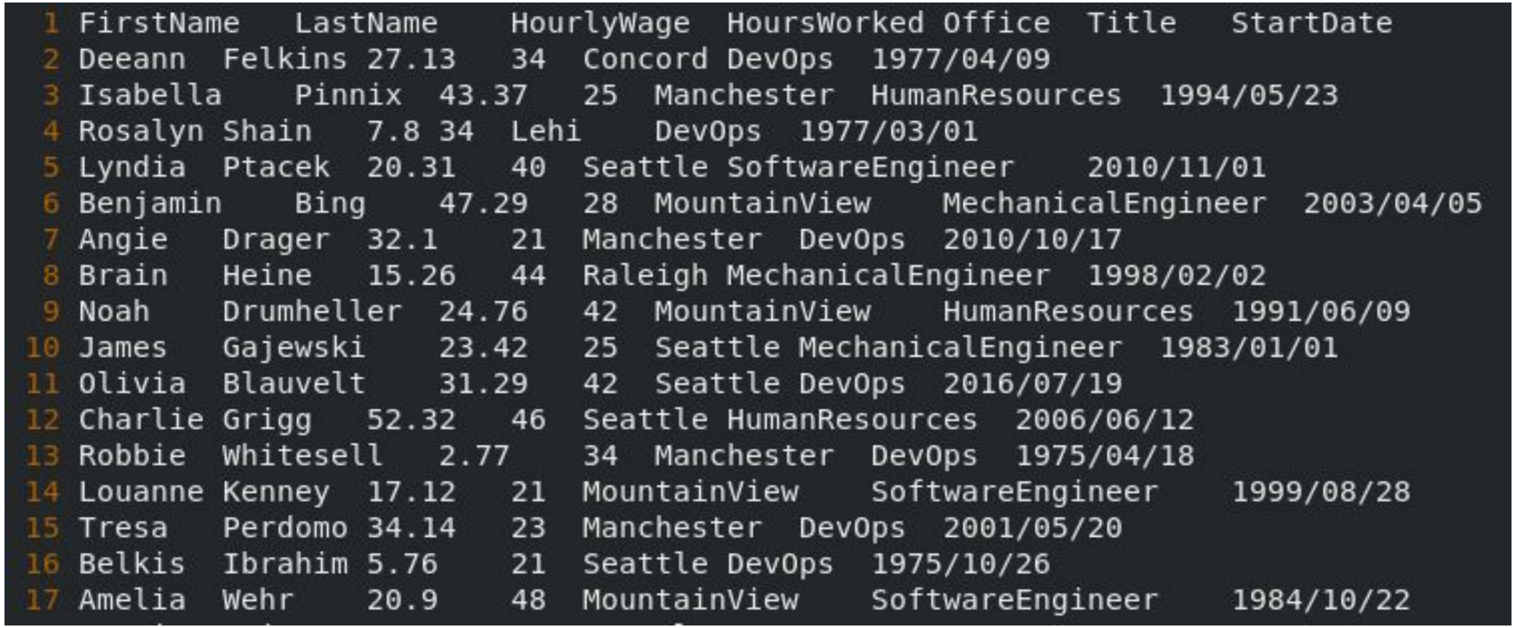
```
#!/usr/bin/env bash

awk '$1 !~ /FirstName/ { print $5 }' payroll.tsv  \
     | sort \
     | uniq \
     | awk 'END { print NR }'
```



```
[ben@localhost awk-hack-the-planet]$ ./09-awk.sh
8
```

Challenges - 10

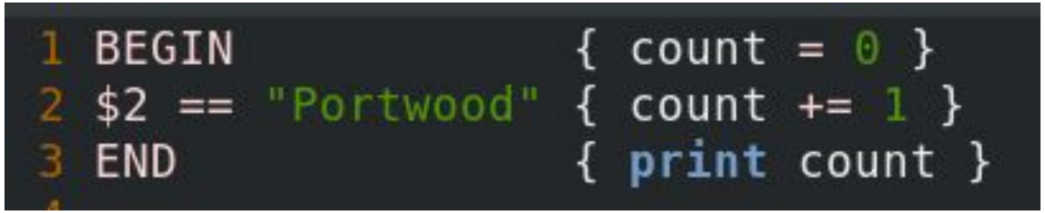
Q. How many people from the Portwood family work here?



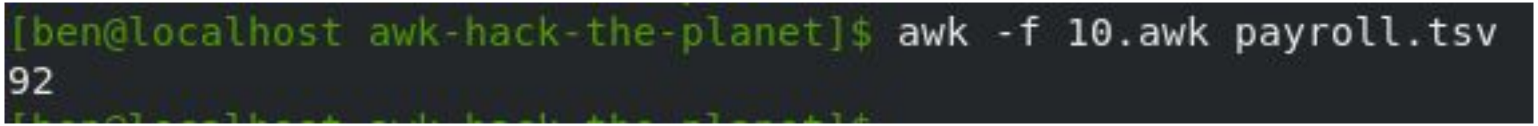
```
FirstName   LastName    HourlyWage  HoursWorked Office  Title   StartDate
Deeann  Felkins 27.13   34  Concord DevOps  1977/04/09
Isabella    Pinnix  43.37   25  Manchester  HumanResources  1994/05/23
Rosalyn Shain   7.8 34  Lehi    DevOps  1977/03/01
Lyndia  Ptacek  20.31   40  Seattle SoftwareEngineer    2010/11/01
Benjamin    Bing    47.29   28  MountainView    MechanicalEngineer  2003/04/05
Angie   Drager  32.1    21  Manchester  DevOps  2010/10/17
Brain   Heine   15.26   44  Raleigh MechanicalEngineer  1998/02/02
Noah    Drumheller  24.76   42  MountainView    HumanResources  1991/06/09
James   Gajewski    23.42   25  Seattle MechanicalEngineer  1983/01/01
Olivia  Blauvelt    31.29   42  Seattle DevOps  2016/07/19
Charlie Grigg   52.32   46  Seattle HumanResources  2006/06/12
Robbie  Whitesell   2.77    34  Manchester  DevOps  1975/04/18
Louanne Kenney  17.12   21  MountainView    SoftwareEngineer    1999/08/28
Tresa   Perdomo 34.14   23  Manchester  DevOps  2001/05/20
Belkis  Ibrahim 5.76    21  Seattle DevOps  1975/10/26
Amelia  Wehr    20.9    48  MountainView    SoftwareEngineer    1984/10/22
```

Challenges - 10

Q. How many people from the Portwood family work here?



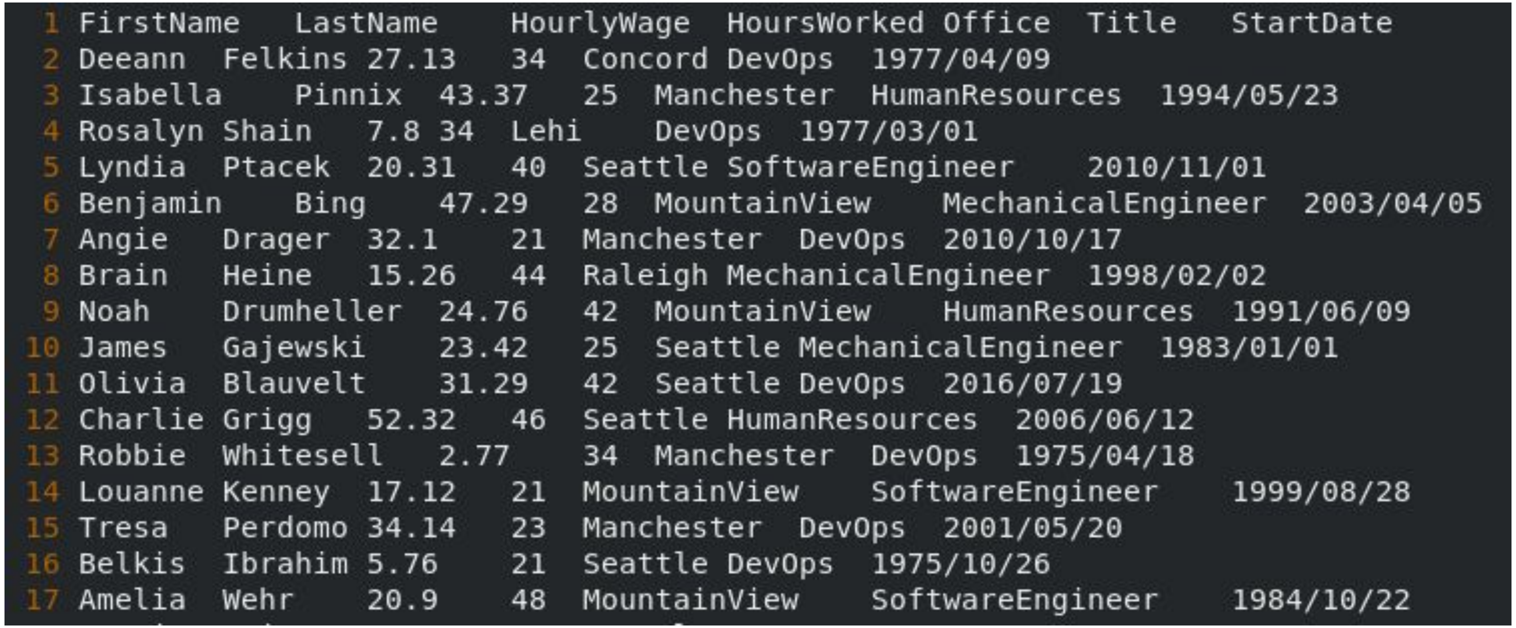
```
BEGIN                  { count = 0 }
$2 == "Portwood"       { count += 1 }
END                    { print count }
```



```
[ben@localhost awk-hack-the-planet]$ awk -f 10.awk payroll.tsv
92
```

Challenges - 11

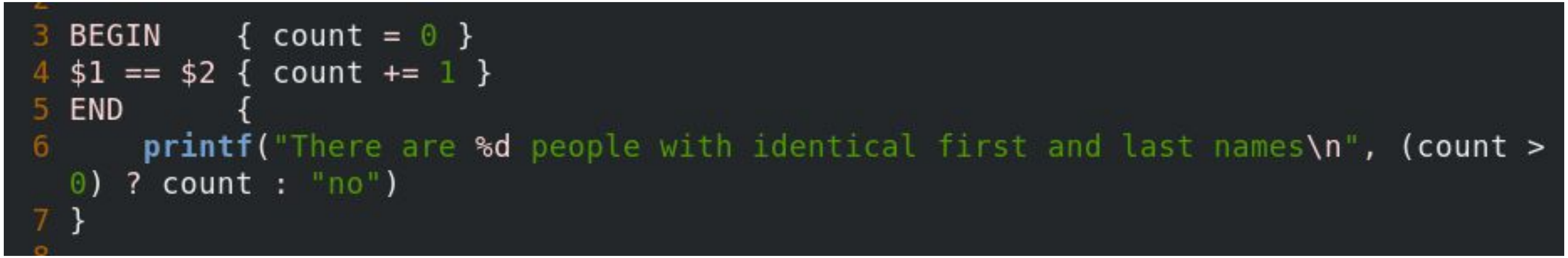
Q. Are there any employees with identical first & last names?



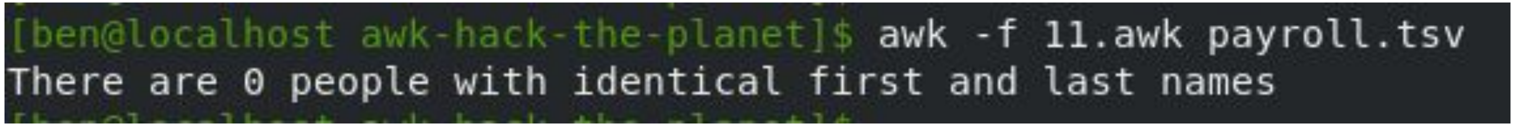
```
FirstName   LastName    HourlyWage  HoursWorked Office  Title   StartDate
Deeann  Felkins 27.13   34  Concord DevOps  1977/04/09
Isabella    Pinnix  43.37   25  Manchester  HumanResources  1994/05/23
Rosalyn Shain   7.8 34  Lehi    DevOps  1977/03/01
Lyndia  Ptacek  20.31   40  Seattle SoftwareEngineer    2010/11/01
Benjamin    Bing    47.29   28  MountainView    MechanicalEngineer  2003/04/05
Angie   Drager  32.1    21  Manchester  DevOps  2010/10/17
Brain   Heine   15.26   44  Raleigh MechanicalEngineer  1998/02/02
Noah    Drumheller  24.76   42  MountainView    HumanResources  1991/06/09
James   Gajewski    23.42   25  Seattle MechanicalEngineer  1983/01/01
Olivia  Blauvelt    31.29   42  Seattle DevOps  2016/07/19
Charlie Grigg   52.32   46  Seattle HumanResources  2006/06/12
Robbie  Whitesell   2.77    34  Manchester  DevOps  1975/04/18
Louanne Kenney  17.12   21  MountainView    SoftwareEngineer    1999/08/28
Tresa   Perdomo 34.14   23  Manchester  DevOps  2001/05/20
Belkis  Ibrahim 5.76    21  Seattle DevOps  1975/10/26
Amelia  Wehr    20.9    48  MountainView    SoftwareEngineer    1984/10/22
```

Challenges - 11

Q. Are there any employees with identical first & last names?



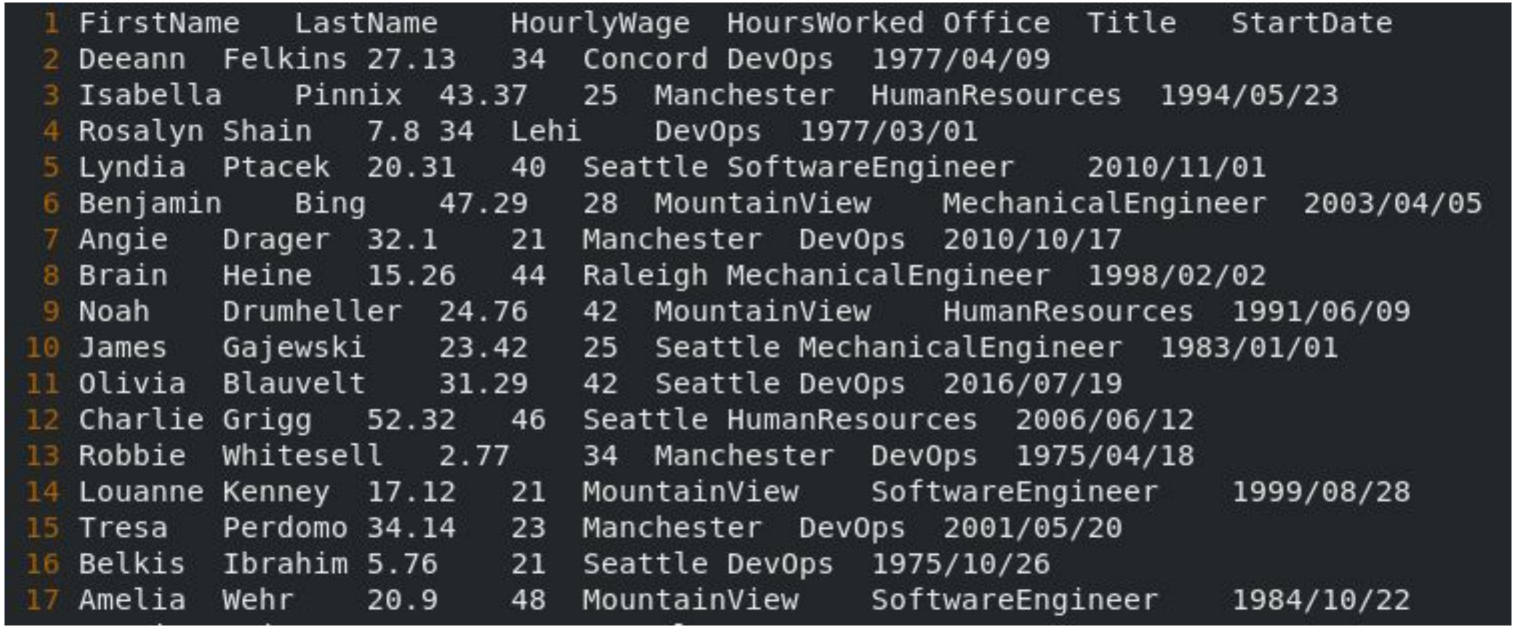
```
BEGIN    { count = 0 }
$1 == $2 { count += 1 }
END      {
    printf("There are %d people with identical first and last names\n", (count >
0) ? count : "no")
}
```



```
[ben@localhost awk-hack-the-planet]$ awk -f 11.awk payroll.tsv
There are 0 people with identical first and last names
```

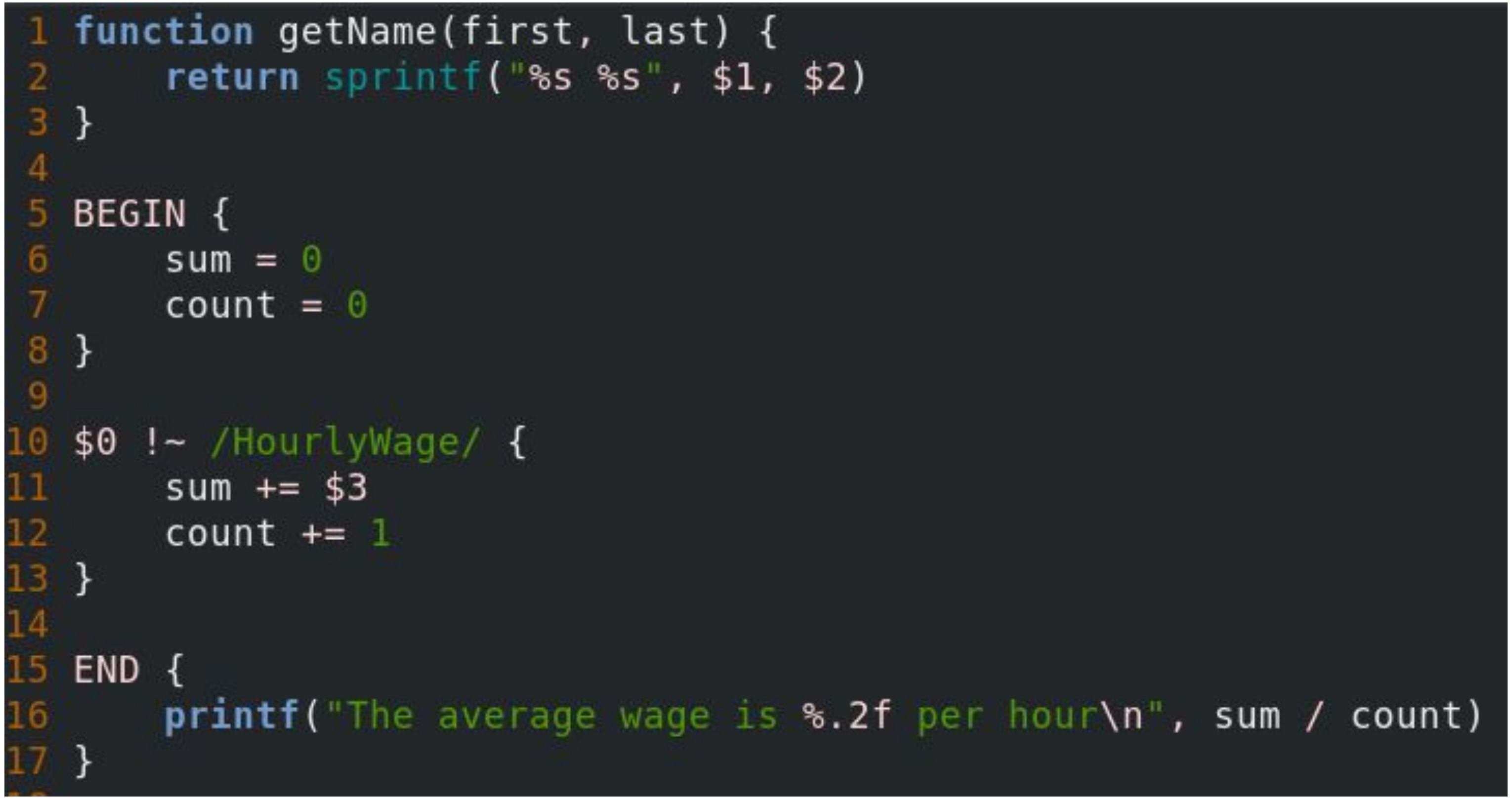
Challenges - 12

Q. What is the average wage?



```
FirstName   LastName    HourlyWage  HoursWorked Office  Title   StartDate
Deeann  Felkins 27.13   34  Concord DevOps  1977/04/09
Isabella    Pinnix  43.37   25  Manchester  HumanResources  1994/05/23
Rosalyn Shain   7.8 34  Lehi    DevOps  1977/03/01
Lyndia  Ptacek  20.31   40  Seattle SoftwareEngineer    2010/11/01
Benjamin    Bing    47.29   28  MountainView    MechanicalEngineer  2003/04/05
Angie   Drager  32.1    21  Manchester  DevOps  2010/10/17
Brain   Heine   15.26   44  Raleigh MechanicalEngineer  1998/02/02
Noah    Drumheller  24.76   42  MountainView    HumanResources  1991/06/09
James   Gajewski    23.42   25  Seattle MechanicalEngineer  1983/01/01
Olivia  Blauvelt    31.29   42  Seattle DevOps  2016/07/19
Charlie Grigg   52.32   46  Seattle HumanResources  2006/06/12
Robbie  Whitesell   2.77    34  Manchester  DevOps  1975/04/18
Louanne Kenney  17.12   21  MountainView    SoftwareEngineer    1999/08/28
Tresa   Perdomo 34.14   23  Manchester  DevOps  2001/05/20
Belkis  Ibrahim 5.76    21  Seattle DevOps  1975/10/26
Amelia  Wehr    20.9    48  MountainView    SoftwareEngineer    1984/10/22
```

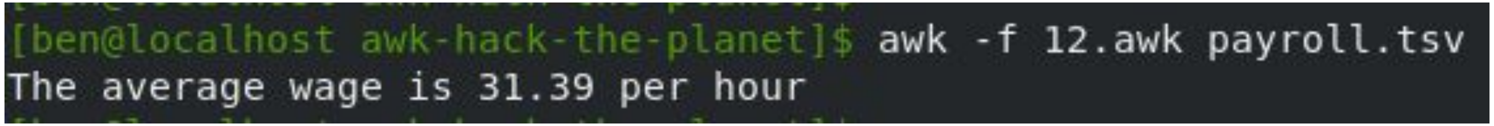
Challenges - 12



```
function getName(first, last) {
    return sprintf("%s %s", $1, $2)
}

BEGIN {
    sum = 0
    count = 0
}

$0 !~ /HourlyWage/ {
    sum += $3
    count += 1
}

END {
    printf("The average wage is %.2f per hour\n", sum / count)
}
```

Challenges - 12

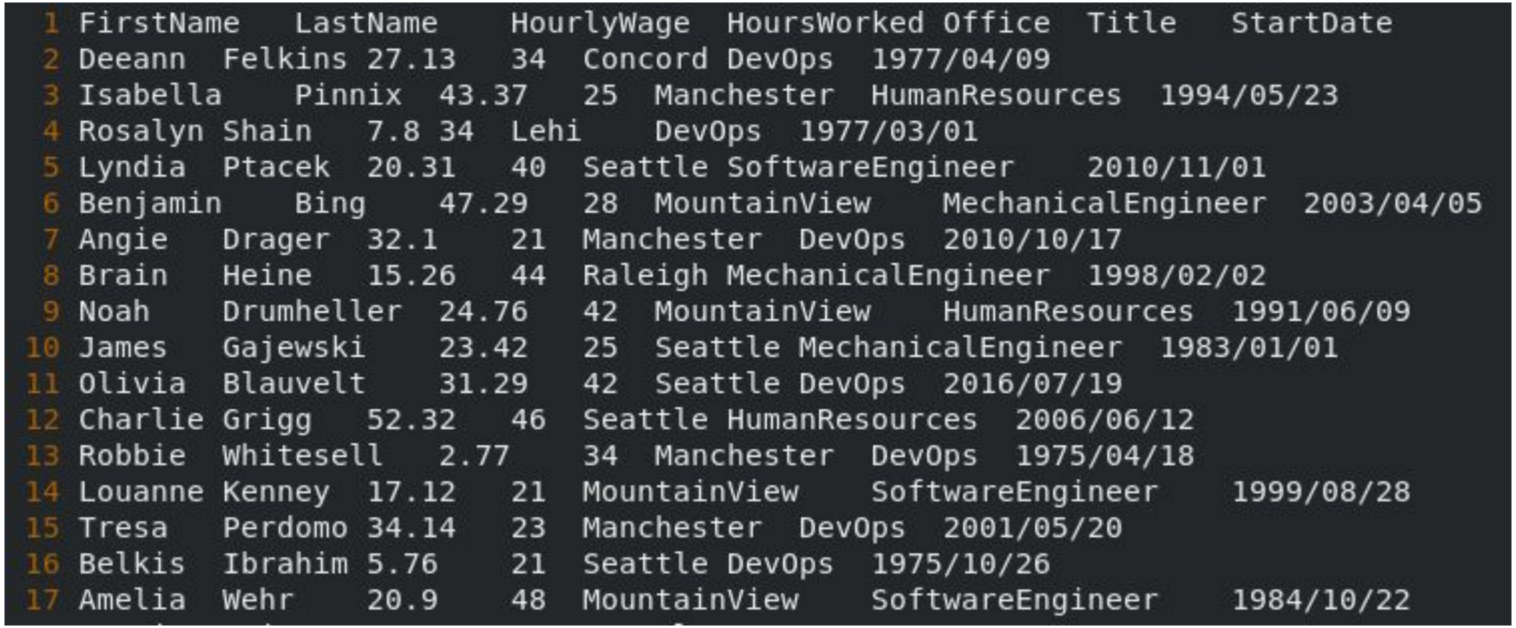
Q. What is the average wage?



```
[ben@localhost awk-hack-the-planet]$ awk -f 12.awk payroll.tsv
The average wage is 31.39 per hour
```

Challenges - 13

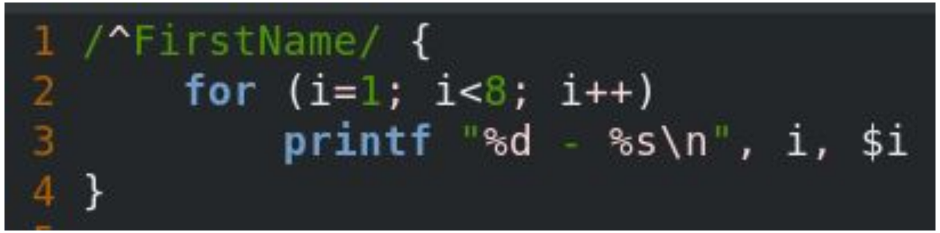
Q. Print each column header, along with which column it is. E.g. The LastName column is the second column, so print "2 - LastName"



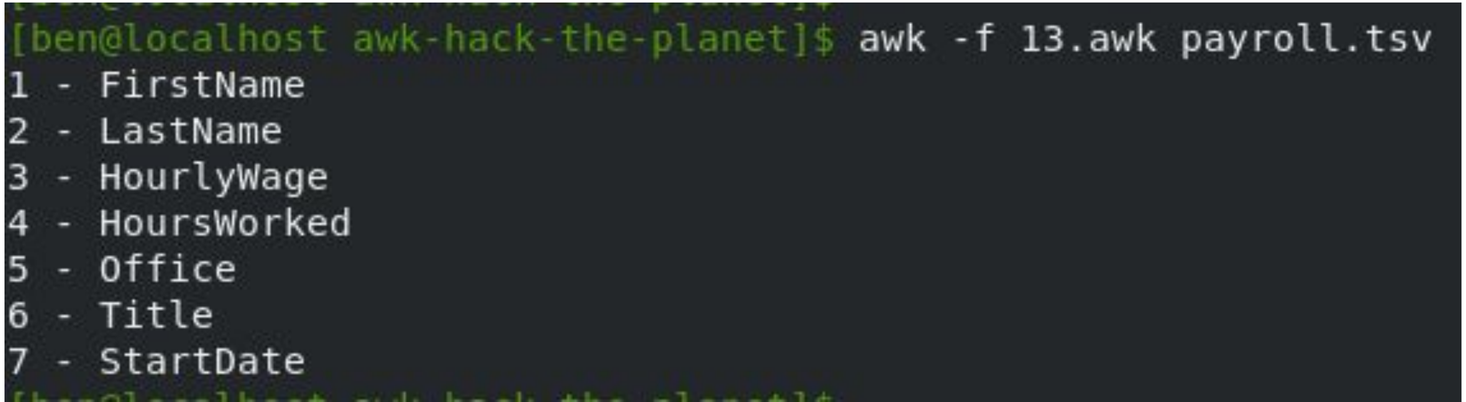
```
FirstName   LastName    HourlyWage  HoursWorked Office  Title   StartDate
Deeann  Felkins 27.13   34  Concord DevOps  1977/04/09
Isabella    Pinnix  43.37   25  Manchester  HumanResources  1994/05/23
Rosalyn Shain   7.8 34  Lehi    DevOps  1977/03/01
Lyndia  Ptacek  20.31   40  Seattle SoftwareEngineer    2010/11/01
Benjamin    Bing    47.29   28  MountainView    MechanicalEngineer  2003/04/05
Angie   Drager  32.1    21  Manchester  DevOps  2010/10/17
Brain   Heine   15.26   44  Raleigh MechanicalEngineer  1998/02/02
Noah    Drumheller  24.76   42  MountainView    HumanResources  1991/06/09
James   Gajewski    23.42   25  Seattle MechanicalEngineer  1983/01/01
Olivia  Blauvelt    31.29   42  Seattle DevOps  2016/07/19
Charlie Grigg   52.32   46  Seattle HumanResources  2006/06/12
Robbie  Whitesell   2.77    34  Manchester  DevOps  1975/04/18
Louanne Kenney  17.12   21  MountainView    SoftwareEngineer    1999/08/28
Tresa   Perdomo 34.14   23  Manchester  DevOps  2001/05/20
Belkis  Ibrahim 5.76    21  Seattle DevOps  1975/10/26
Amelia  Wehr    20.9    48  MountainView    SoftwareEngineer    1984/10/22
```

Challenges - 13

Q. Print each column header, along with which column it is. E.g. The LastName column is the second column, so print "2 - LastName"



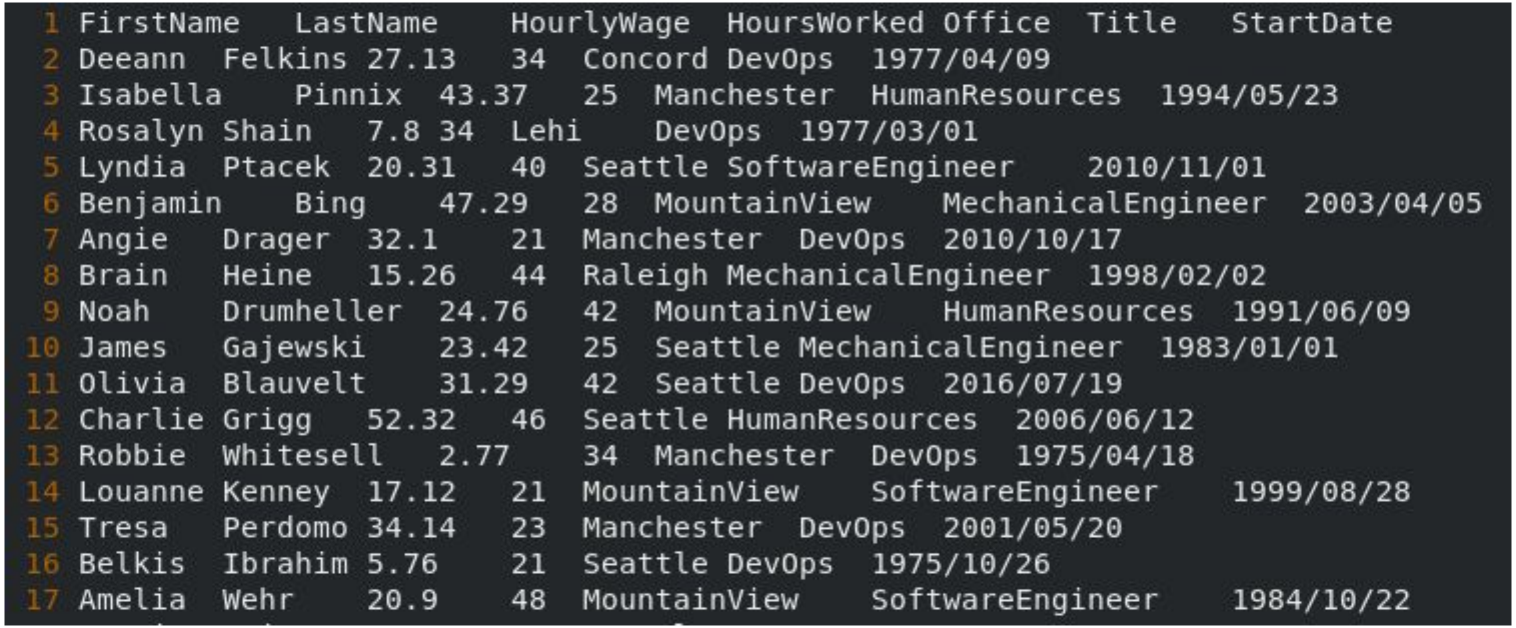
```
/^FirstName/ {
    for (i=1; i<8; i++)
        printf "%d - %s\n", i, $i
}
```



```
[ben@localhost awk-hack-the-planet]$ awk -f 13.awk payroll.tsv
1 - FirstName
2 - LastName
3 - HourlyWage
4 - HoursWorked
5 - Office
6 - Title
7 - StartDate
```

Challenges - 14

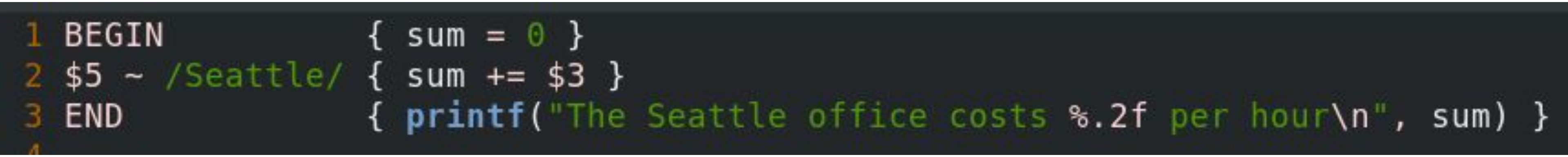
Q. How much money per hour does the Seattle office cost?



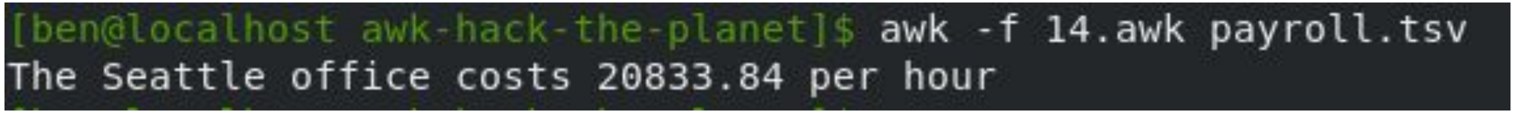
```
FirstName   LastName    HourlyWage  HoursWorked Office  Title   StartDate
Deeann  Felkins 27.13   34  Concord DevOps  1977/04/09
Isabella    Pinnix  43.37   25  Manchester  HumanResources  1994/05/23
Rosalyn Shain   7.8 34  Lehi    DevOps  1977/03/01
Lyndia  Ptacek  20.31   40  Seattle SoftwareEngineer    2010/11/01
Benjamin    Bing    47.29   28  MountainView    MechanicalEngineer  2003/04/05
Angie   Drager  32.1    21  Manchester  DevOps  2010/10/17
Brain   Heine   15.26   44  Raleigh MechanicalEngineer  1998/02/02
Noah    Drumheller  24.76   42  MountainView    HumanResources  1991/06/09
James   Gajewski    23.42   25  Seattle MechanicalEngineer  1983/01/01
Olivia  Blauvelt    31.29   42  Seattle DevOps  2016/07/19
Charlie Grigg   52.32   46  Seattle HumanResources  2006/06/12
Robbie  Whitesell   2.77    34  Manchester  DevOps  1975/04/18
Louanne Kenney  17.12   21  MountainView    SoftwareEngineer    1999/08/28
Tresa   Perdomo 34.14   23  Manchester  DevOps  2001/05/20
Belkis  Ibrahim 5.76    21  Seattle DevOps  1975/10/26
Amelia  Wehr    20.9    48  MountainView    SoftwareEngineer    1984/10/22
```

Challenges - 14

Q. How much money per hour does the Seattle office cost?



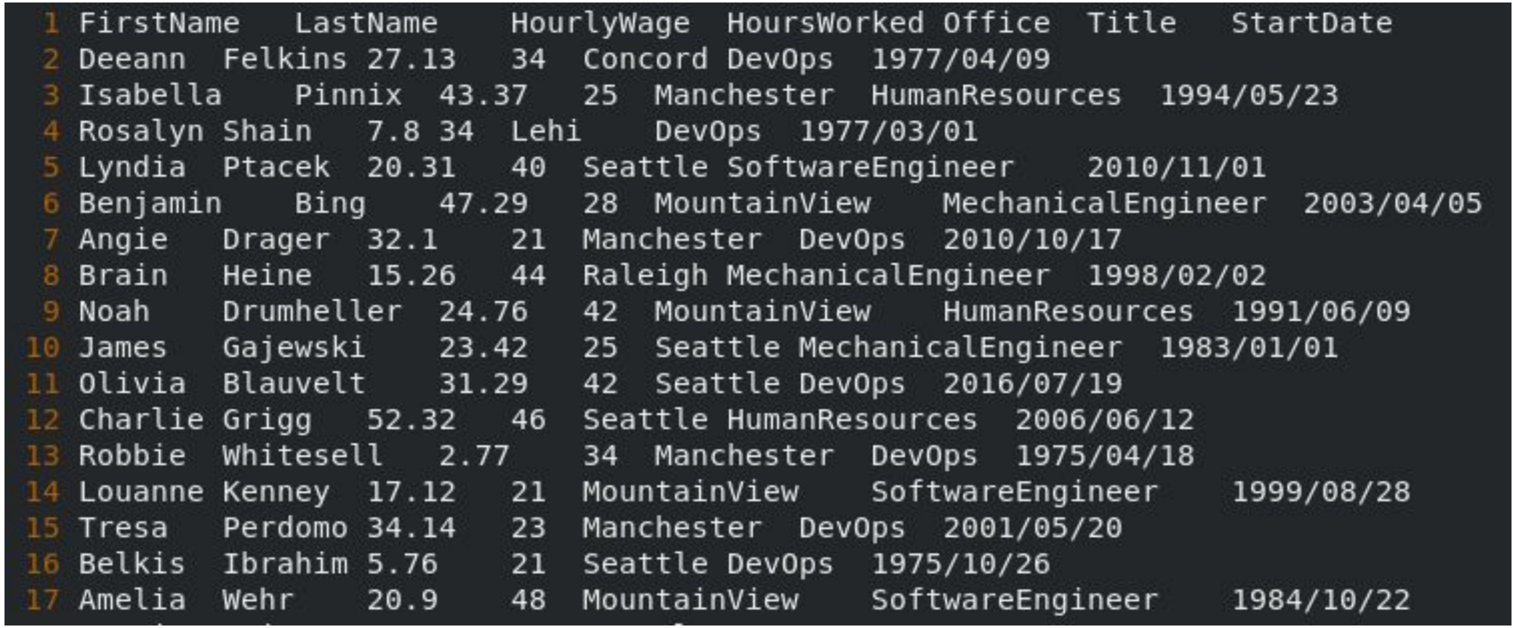
```
BEGIN               { sum = 0 }
$5 ~ /Seattle/ { sum += $3 }
END                 { printf("The Seattle office costs %.2f per hour\n", sum) }
```



```
[ben@localhost awk-hack-the-planet]$ awk -f 14.awk payroll.tsv
The Seattle office costs 20833.84 per hour
```

Challenges - 15

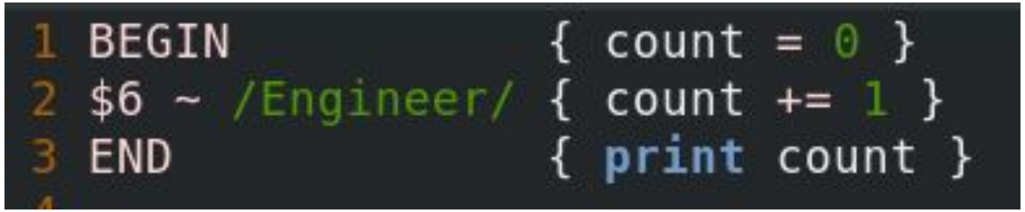
Q. How many engineers (of any type) work here?



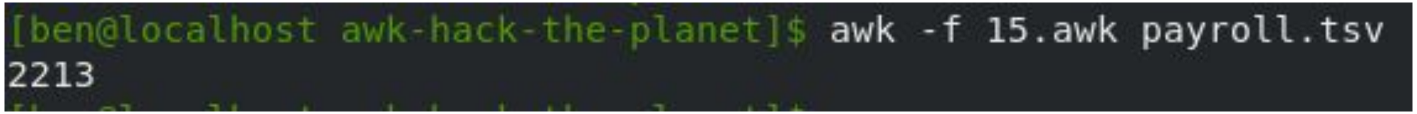
```
FirstName   LastName    HourlyWage  HoursWorked Office  Title   StartDate
Deeann  Felkins 27.13   34  Concord DevOps  1977/04/09
Isabella    Pinnix  43.37   25  Manchester  HumanResources  1994/05/23
Rosalyn Shain   7.8 34  Lehi    DevOps  1977/03/01
Lyndia  Ptacek  20.31   40  Seattle SoftwareEngineer    2010/11/01
Benjamin    Bing    47.29   28  MountainView    MechanicalEngineer  2003/04/05
Angie   Drager  32.1    21  Manchester  DevOps  2010/10/17
Brain   Heine   15.26   44  Raleigh MechanicalEngineer  1998/02/02
Noah    Drumheller  24.76   42  MountainView    HumanResources  1991/06/09
James   Gajewski    23.42   25  Seattle MechanicalEngineer  1983/01/01
Olivia  Blauvelt    31.29   42  Seattle DevOps  2016/07/19
Charlie Grigg   52.32   46  Seattle HumanResources  2006/06/12
Robbie  Whitesell   2.77    34  Manchester  DevOps  1975/04/18
Louanne Kenney  17.12   21  MountainView    SoftwareEngineer    1999/08/28
Tresa   Perdomo 34.14   23  Manchester  DevOps  2001/05/20
Belkis  Ibrahim 5.76    21  Seattle DevOps  1975/10/26
Amelia  Wehr    20.9    48  MountainView    SoftwareEngineer    1984/10/22
```

Challenges - 15

Q. How many engineers (of any type) work here?



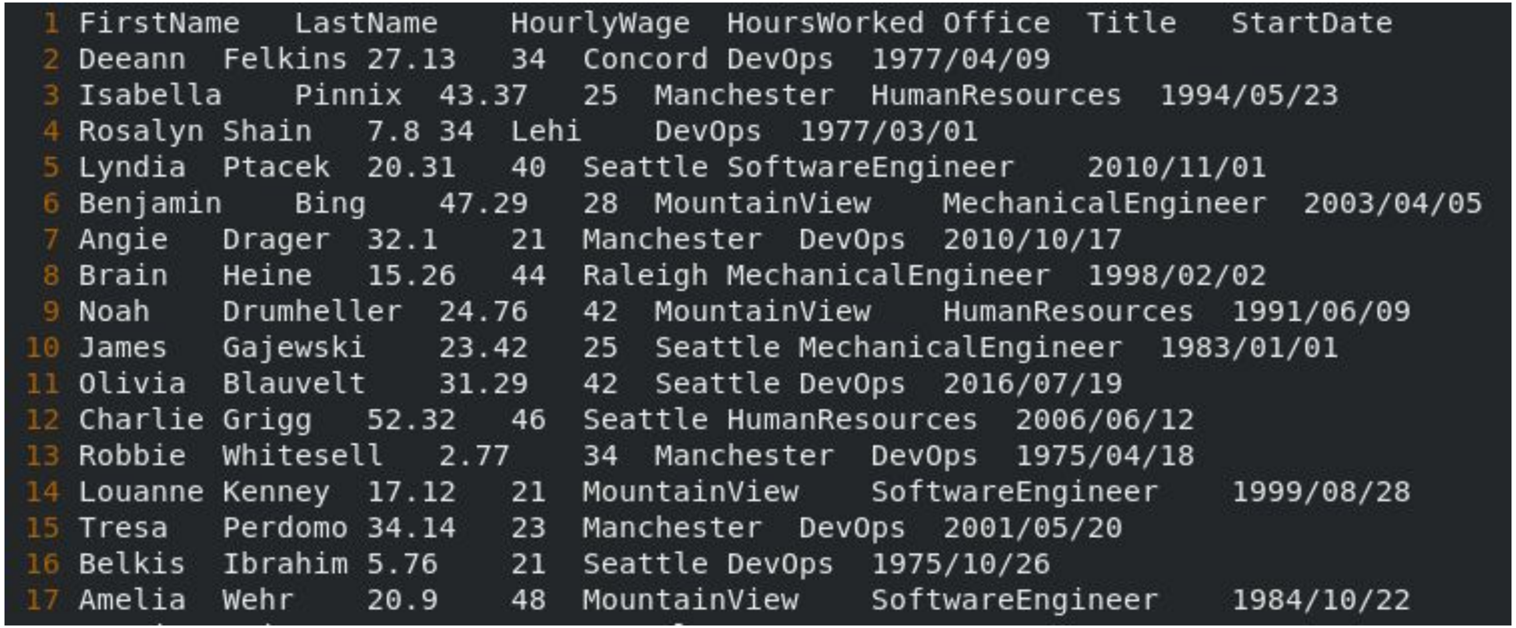
```
BEGIN                { count = 0 }
$6 ~ /Engineer/ { count += 1 }
END                  { print count }
```



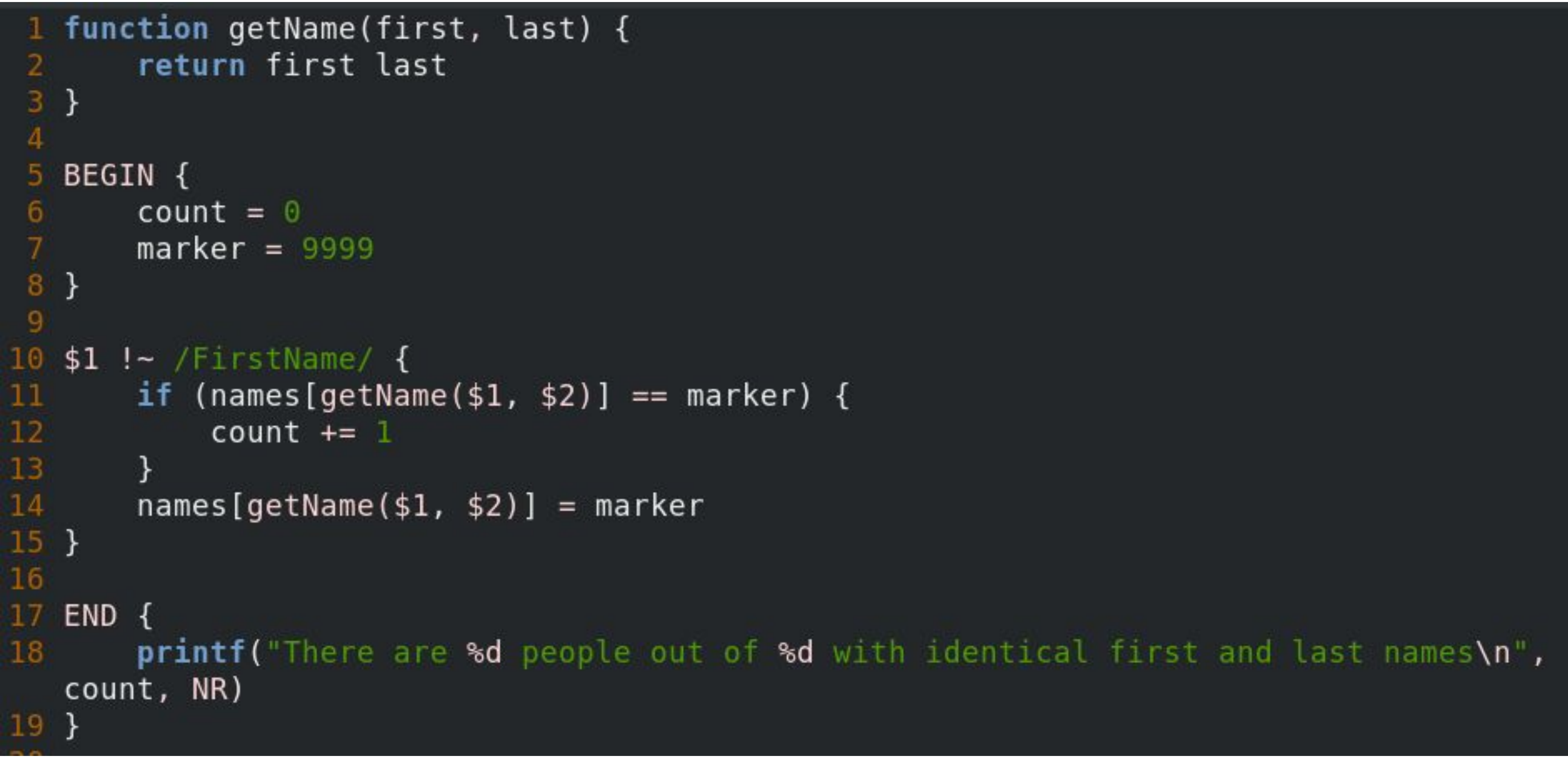
```
[ben@localhost awk-hack-the-planet]$ awk -f 15.awk payroll.tsv
2213
```

Challenges - 16

Q. Are there any duplicate entries? (Same names appear more than once)



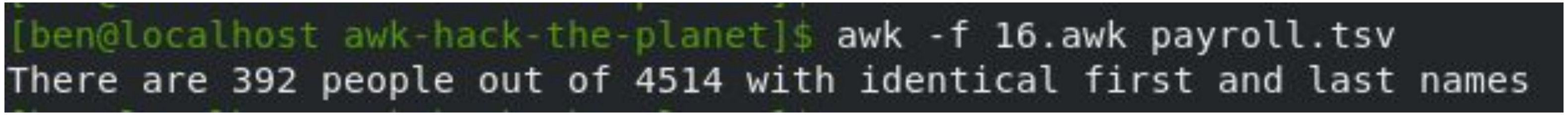
```
FirstName   LastName    HourlyWage  HoursWorked Office  Title   StartDate
Deeann  Felkins 27.13   34  Concord DevOps  1977/04/09
Isabella    Pinnix  43.37   25  Manchester  HumanResources  1994/05/23
Rosalyn Shain   7.8 34  Lehi    DevOps  1977/03/01
Lyndia  Ptacek  20.31   40  Seattle SoftwareEngineer    2010/11/01
Benjamin    Bing    47.29   28  MountainView    MechanicalEngineer  2003/04/05
Angie   Drager  32.1    21  Manchester  DevOps  2010/10/17
Brain   Heine   15.26   44  Raleigh MechanicalEngineer  1998/02/02
Noah    Drumheller  24.76   42  MountainView    HumanResources  1991/06/09
James   Gajewski    23.42   25  Seattle MechanicalEngineer  1983/01/01
Olivia  Blauvelt    31.29   42  Seattle DevOps  2016/07/19
Charlie Grigg   52.32   46  Seattle HumanResources  2006/06/12
Robbie  Whitesell   2.77    34  Manchester  DevOps  1975/04/18
Louanne Kenney  17.12   21  MountainView    SoftwareEngineer    1999/08/28
Tresa   Perdomo 34.14   23  Manchester  DevOps  2001/05/20
Belkis  Ibrahim 5.76    21  Seattle DevOps  1975/10/26
Amelia  Wehr    20.9    48  MountainView    SoftwareEngineer    1984/10/22
```



```
function getName(first, last) {
    return first last
}

BEGIN {
    count = 0
    marker = 9999
}

$1 !~ /FirstName/ {
    if (names[getName($1, $2)] == marker) {
        count += 1
    }
    names[getName($1, $2)] = marker
}

END {
    printf("There are %d people out of %d with identical first and last names\n", count, NR)
}
```

Challenges - 16

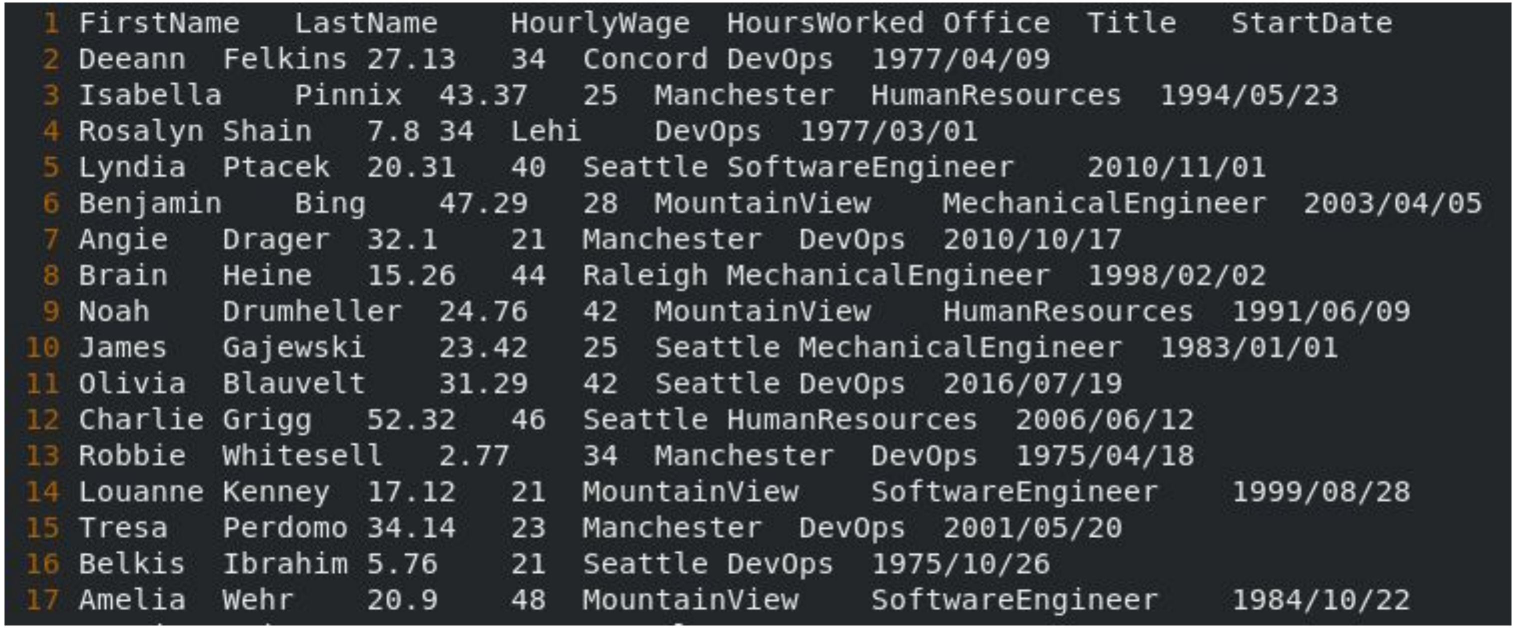
Q. Are there any duplicate entries? (Same names appear more than once)



```
[ben@localhost awk-hack-the-planet]$ awk -f 16.awk payroll.tsv
There are 392 people out of 4514 with identical first and last names
```

Challenges - 17

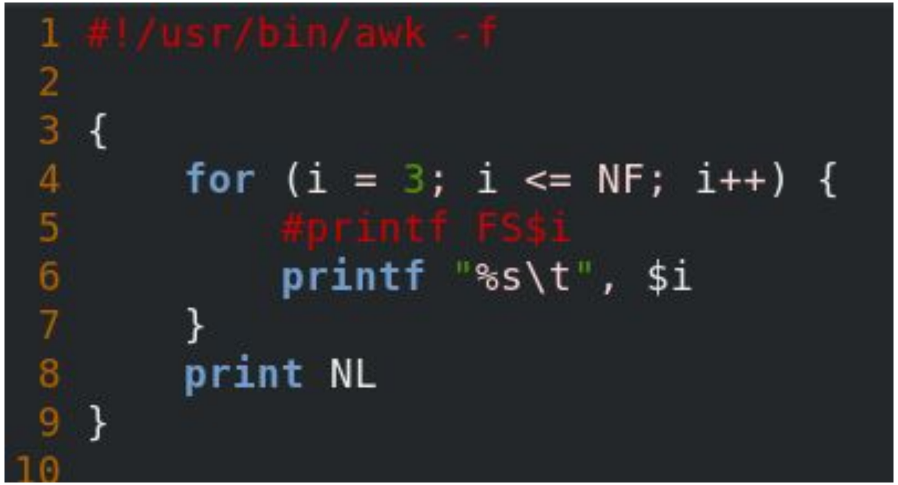
Q. Anonymize the data by removing the first two columns. Print all remaining columns



```
FirstName   LastName    HourlyWage  HoursWorked Office  Title   StartDate
Deeann  Felkins 27.13   34  Concord DevOps  1977/04/09
Isabella    Pinnix  43.37   25  Manchester  HumanResources  1994/05/23
Rosalyn Shain   7.8 34  Lehi    DevOps  1977/03/01
Lyndia  Ptacek  20.31   40  Seattle SoftwareEngineer    2010/11/01
Benjamin    Bing    47.29   28  MountainView    MechanicalEngineer  2003/04/05
Angie   Drager  32.1    21  Manchester  DevOps  2010/10/17
Brain   Heine   15.26   44  Raleigh MechanicalEngineer  1998/02/02
Noah    Drumheller  24.76   42  MountainView    HumanResources  1991/06/09
James   Gajewski    23.42   25  Seattle MechanicalEngineer  1983/01/01
Olivia  Blauvelt    31.29   42  Seattle DevOps  2016/07/19
Charlie Grigg   52.32   46  Seattle HumanResources  2006/06/12
Robbie  Whitesell   2.77    34  Manchester  DevOps  1975/04/18
Louanne Kenney  17.12   21  MountainView    SoftwareEngineer    1999/08/28
Tresa   Perdomo 34.14   23  Manchester  DevOps  2001/05/20
Belkis  Ibrahim 5.76    21  Seattle DevOps  1975/10/26
Amelia  Wehr    20.9    48  MountainView    SoftwareEngineer    1984/10/22
```

Challenges - 17

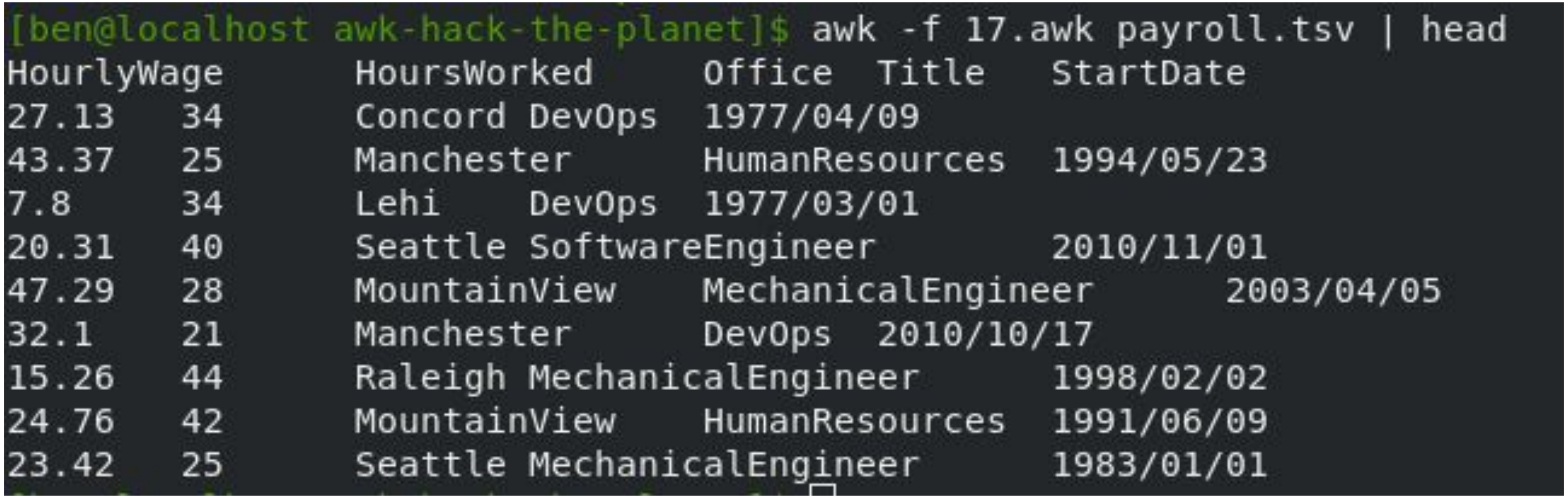
Q. Anonymize the data by removing the first two columns. Print all remaining columns



```
#!/usr/bin/awk -f

{
	for (i = 3; i <= NF; i++) {
		#printf FS$i
		printf "%s\t", $i
	}
	print NL
}

```

Challenges - 17

Q. Anonymize the data by removing the first two columns. Print all remaining columns



```
[ben@localhost awk-hack-the-planet]$ awk -f 17.awk payroll.tsv | head
HourlyWage      HoursWorked     Office  Title   StartDate
27.13   34      Concord DevOps  1977/04/09
43.37   25      Manchester      HumanResources  1994/05/23
7.8     34      Lehi    DevOps  1977/03/01
20.31   40      Seattle SoftwareEngineer        2010/11/01
47.29   28      MountainView    MechanicalEngineer      2003/04/05
32.1    21      Manchester      DevOps  2010/10/17
15.26   44      Raleigh MechanicalEngineer      1998/02/02
24.76   42      MountainView    HumanResources  1991/06/09
23.42   25      Seattle MechanicalEngineer      1983/01/01
```