Computing Unit 3: Data Types

WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

Kurt Hornik

September 26, 2018

EFMD EQUIS ACCREDITED AACSB ACCREDITED ASSOCIATION OF MBAS ACCREDITED

Character vectors

- String constants:
  - enclosed in `"..."` (double quotes), alternatively single quotes.

Character vectors

- String constants:
  - enclosed in `"..."` (double quotes), alternatively single quotes.
  - C-style special characters in double-quoted string constants:

    ```
    \a   \n   \t   \\   \"
    ```

    etc.

Character vectors

- String constants:
  - enclosed in `"..."` (double quotes), alternatively single quotes.
  - C-style special characters in double-quoted string constants:

    ```
    \a   \n   \t   \\   \"
    ```

    etc.
- Combination using `c()`.

Character vectors

- String constants:
  - enclosed in `"..."` (double quotes), alternatively single quotes.
  - C-style special characters in double-quoted string constants:

    ```
    \a   \n   \t   \\   \"
    ```

    etc.
- Combination using `c()`.
- Elementwise equality using `==`

Character vectors

- String constants:
  - enclosed in `"..."` (double quotes), alternatively single quotes.
  - C-style special characters in double-quoted string constants:

    ```
    \a   \n   \t   \\   \"
    ```

    etc.
- Combination using `c()`.
- Elementwise equality using `==`
- Creation and testing of character vectors using `character` et al.

Character vectors

`nchar` count the number of characters

Character vectors

`nchar`	count the number of characters
`substr`, `substring`	extract or replace substrings

Character vectors

`nchar`	count the number of characters
`substr`, `substring`	extract or replace substrings
`paste`	concatenate vectors after converting to character

Character vectors

`nchar`	count the number of characters
`substr`, `substring`	extract or replace substrings
`paste`	concatenate vectors after converting to character
`sprintf`	C-style string formatting

Character vectors

`nchar`	count the number of characters
`substr`, `substring`	extract or replace substrings
`paste`	concatenate vectors after converting to character
`sprintf`	C-style string formatting
`iconv`	convert between encodings

Character vectors

`nchar`	count the number of characters
`substr`, `substring`	extract or replace substrings
`paste`	concatenate vectors after converting to character
`sprintf`	C-style string formatting
`iconv`	convert between encodings
`chartr`, `toupper`, `tolower`	character translation and case folding

Character vectors

`match`, `pmatch` (simple) complete and partial matching of character strings

Character vectors

`match`, `pmatch` (simple) complete and partial matching of character strings

`grep`, `grepl`, `sub`, `gsub`, `regexpr`, `gregexpr` pattern matching and replacement using regular expressions

Character vectors

`match`, `pmatch`	(simple) complete and partial matching of character strings
`grep`, `grepl`, `sub`, `gsub`, `regexpr`, `gregexpr`	pattern matching and replacement using regular expressions
`strsplit`	split strings into substrings

Regular Expressions

- A “regular expression” (“regexp”, “RE”) is a pattern that denotes a (possibly infinite) set of strings.

Regular Expressions

- A “regular expression” (“regexp”, “RE”) is a pattern that denotes a (possibly infinite) set of strings.
- Portable Operating System Interface (POSIX) 1003.2 defines modern “extended” regexps, and obsolete “basic” regexps.

Regular Expressions

- A “regular expression” (“regexp”, “RE”) is a pattern that denotes a (possibly infinite) set of strings.
- Portable Operating System Interface (POSIX) 1003.2 defines modern “extended” regexps, and obsolete “basic” regexps.
- REs have a syntax in which a few characters are special constructs and the rest are “ordinary”.

Regular Expressions

- A “regular expression” (“regexp”, “RE”) is a pattern that denotes a (possibly infinite) set of strings.
- Portable Operating System Interface (POSIX) 1003.2 defines modern “extended” regexps, and obsolete “basic” regexps.
- REs have a syntax in which a few characters are special constructs and the rest are “ordinary”.
- An ordinary character is a simple regexp matching just itself.

Regular Expressions

- A “regular expression” (“regexp”, “RE”) is a pattern that denotes a (possibly infinite) set of strings.
- Portable Operating System Interface (POSIX) 1003.2 defines modern “extended” regexps, and obsolete “basic” regexps.
- REs have a syntax in which a few characters are special constructs and the rest are “ordinary”.
- An ordinary character is a simple regexp matching just itself.
- ERE special characters: `. * + ? ^ $ | \ [ ] ( ) { }`.

Regular Expressions

- A “regular expression” (“regexp”, “RE”) is a pattern that denotes a (possibly infinite) set of strings.
- Portable Operating System Interface (POSIX) 1003.2 defines modern “extended” regexps, and obsolete “basic” regexps.
- REs have a syntax in which a few characters are special constructs and the rest are “ordinary”.
- An ordinary character is a simple regexp matching just itself.
- ERE special characters: `. * + ? ^ $ | \ [ ] ( ) { }`.
- Any non-special character is ordinary unless preceded by a \.

Regexp Special Characters

. Matches any single character.

Regexp Special Characters

- `.` Matches any single character.
- `*` Postfix operator: matches the preceding regexp as many times as possible (zero or more).

Regexp Special Characters

.	Matches any single character.
*	Postfix operator: matches the preceding regexp as many times as possible (zero or more).
+	Postfix operator: matches the preceding regexp as many times as possible, but at least once.

Regexp Special Characters

- `.` Matches any single character.
- `*` Postfix operator: matches the preceding regexp as many times as possible (zero or more).
- `+` Postfix operator: matches the preceding regexp as many times as possible, but at least once.
- `?` Postfix operator: matches the preceding regexp once or not at all.

Regexp Special Characters

.	Matches any single character.
*	Postfix operator: matches the preceding regexp as many times as possible (zero or more).
+	Postfix operator: matches the preceding regexp as many times as possible, but at least once.
?	Postfix operator: matches the preceding regexp once or not at all.
^	Matches the null string at the beginning of a line

Regexp Special Characters

.	Matches any single character.
*	Postfix operator: matches the preceding regexp as many times as possible (zero or more).
+	Postfix operator: matches the preceding regexp as many times as possible, but at least once.
?	Postfix operator: matches the preceding regexp once or not at all.
^	Matches the null string at the beginning of a line
$	Matches the null string at the end of a line

Regexp Special Characters

.	Matches any single character.
*	Postfix operator: matches the preceding regexp as many times as possible (zero or more).
+	Postfix operator: matches the preceding regexp as many times as possible, but at least once.
?	Postfix operator: matches the preceding regexp once or not at all.
^	Matches the null string at the beginning of a line
$	Matches the null string at the end of a line
|	specifies an alternative (applies to the largest possible surrounding).

Regexp Special Characters

- `.` Matches any single character.
- `*` Postfix operator: matches the preceding regexp as many times as possible (zero or more).
- `+` Postfix operator: matches the preceding regexp as many times as possible, but at least once.
- `?` Postfix operator: matches the preceding regexp once or not at all.
- `^` Matches the null string at the beginning of a line
- `$` Matches the null string at the end of a line
- `|` specifies an alternative (applies to the largest possible surrounding).
- `\` Quotes special characters, and introduces additional special constructs.

Regexp Character Alternatives

- `[ ... ]` is a character alternative (bracket expression) matching one of the specified characters.

Regexp Character Alternatives

- `[ ... ]` is a character alternative (bracket expression) matching one of the specified characters.
- Character ranges can be included by writing the starting and ending characters with a `-` between them.

Regexp Character Alternatives

- `[ ... ]` is a character alternative (bracket expression) matching one of the specified characters.
- Character ranges can be included by writing the starting and ending characters with a `-` between them.
- Inside character alternatives, the following are special: `]` `-` `^`.

Regexp Character Alternatives

- `[ ... ]` is a character alternative (bracket expression) matching one of the specified characters.
- Character ranges can be included by writing the starting and ending characters with a `-` between them.
- Inside character alternatives, the following are special: `]` `-` `^`.
- `[^ ... ]` is a complementary character alternative matching any character except the ones specified.

Regexp Character Alternatives

- `[ ... ]` is a character alternative (bracket expression) matching one of the specified characters.
- Character ranges can be included by writing the starting and ending characters with a `-` between them.
- Inside character alternatives, the following are special: `]` `-` `^`.
- `[^ ... ]` is a complementary character alternative matching any character except the ones specified.
- Inside a character alternative, one can also use character classes by enclosing their names in `[: ... :]` (character classes are alnum, alpha, blank, cntrl, digit, graph, lower, print, punct, space, upper, xdigit).

Regexp Grouping

(. . .) specifies a grouping. Used

1. to enclose a set of | alternatives;

Regexp Grouping

(. . .) specifies a grouping. Used

1. to enclose a set of | alternatives;
2. to enclose a complicated expression for the postfix operators;

Regexp Grouping

`( ... )` specifies a grouping. Used

1. to enclose a set of | alternatives;
2. to enclose a complicated expression for the postfix operators;
3. record a matched substring for future reference with \DIGIT

Regexp Bounds

`{ ... }` specifies a bound on the preceding regexp (atom).

Regexp Bounds

`{ ... }` specifies a bound on the preceding regexp (atom).

1. $\{m\}$, $0 \leq m \leq 255$, matches a sequence of exactly m repetitions of the preceding regexp.

Regexp Bounds

`{ ... }` specifies a bound on the preceding regexp (atom).

1. $\{m\}$, $0 \le m \le 255$, matches a sequence of exactly m repetitions of the preceding regexp.
2. $\{m,\}$, $0 \le m \le 255$, matches a sequence of at least m repetitions of the preceding regexp.

Regexp Bounds

`{ ... }` specifies a bound on the preceding regexp (atom).

1. $\{m\}$, $0 \le m \le 255$, matches a sequence of exactly m repetitions of the preceding regexp.
2. $\{m,\}$, $0 \le m \le 255$, matches a sequence of at least m repetitions of the preceding regexp.
3. $\{m,n\}$, $0 \le m \le n \le 255$, matches a sequence of m through n (inclusive) repetitions of the preceding regexp.