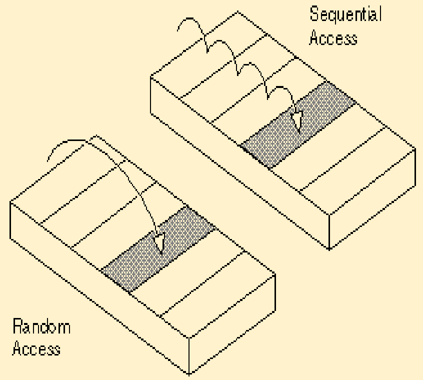
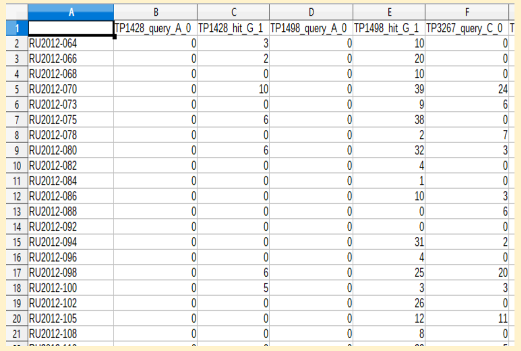
Lecture 5: Reading and writing data

CPSC 499, Fall 2018

What is a file?

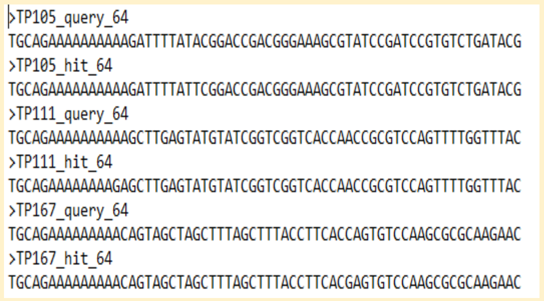


- Data recorded in a storage medium, such as a hard disk, flash drive, or DVD
  - In contrast to objects in the R environment, which are in the computer's RAM
- Always stored as a linear set of bytes (byte = 8 bits, and a bit is a 0 or 1)
- A file has a beginning and an end, and is usually read or written linearly from beginning to end (**sequential access**)
- It is also possible for a file to be **random access**, meaning you can jump to any point in the file. We will do some random access later with Bioconductor.



	A	B	C	D	E	F
1		TP1428_query_A_0	TP1428_hit_G_1	TP1498_query_A_0	TP1498_hit_G_1	TP3267_query_C_0
2	RU2012-064	0	3	0	10	0
3	RU2012-066	0	2	0	20	0
4	RU2012-068	0	0	0	10	0
5	RU2012-070	0	10	0	39	24
6	RU2012-073	0	0	0	9	6
7	RU2012-075	0	6	0	38	0
8	RU2012-078	0	0	0	2	7
9	RU2012-080	0	6	0	32	3
10	RU2012-082	0	0	0	4	0
11	RU2012-084	0	0	0	1	0
12	RU2012-086	0	0	0	10	3
13	RU2012-088	0	0	0	0	6
14	RU2012-092	0	0	0	0	0
15	RU2012-094	0	0	0	31	2
16	RU2012-096	0	0	0	4	0
17	RU2012-098	0	6	0	25	20
18	RU2012-100	0	5	0	3	3
19	RU2012-102	0	0	0	26	0
20	RU2012-105	0	0	0	12	11
21	RU2012-108	0	0	0	8	0

Text files



```
>TP105_query_64
TGCAGAAAAAAAAAAGATTTTATACGGACCGACGGGAAAGCGTATCCGATCCGTGTCTGATACG
>TP105_hit_64
TGCAGAAAAAAAAAAGATTTTATTCGGACCGACGGGAAAGCGTATCCGATCCGTGTCTGATACG
>TP111_query_64
TGCAGAAAAAAAAAAGCTTGAGTATGTATCGGTCGGTCACCAACCGCGTCCAGTTTTGGTTTAC
>TP111_hit_64
TGCAGAAAAAAAAGAGCTTGAGTATGTATCGGTCGGTCACCAACCGCGTCCAGTTTTGGTTTAC
>TP167_query_64
TGCAGAAAAAAAAACAGTAGCTAGCTTTAGCTTTACCTTCACCAGTGTCCAAGCGCGCAAGAAC
>TP167_hit_64
TGCAGAAAAAAAAACAGTAGCTAGCTTTAGCTTTACCTTCACGAGTGTCCAAGCGCGCAAGAAC
```

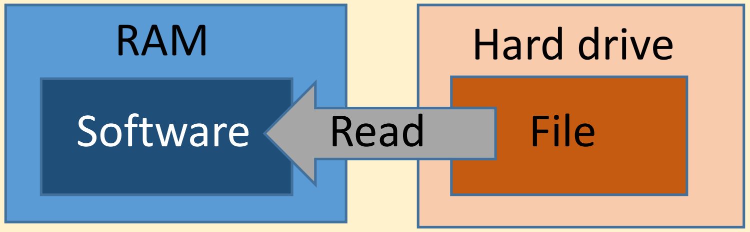
- All the files that we will read and write in R are text files, including spreadsheet-like data such as CSV (comma-separated value) or tab-delimited text
- Text-based formats are generally preferred by scientists
  - Don't require proprietary software to open them
  - Don't require advanced programming skill to process them
  - Good for long-term preservation of data

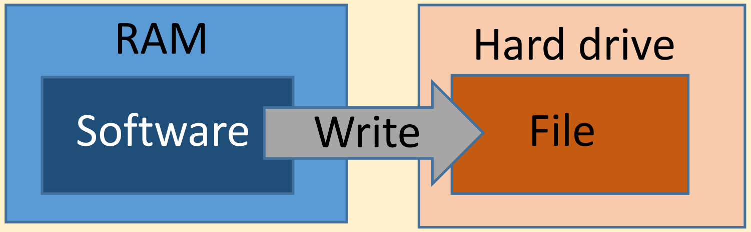
Format of text files

- ASCII is the old way of storing text, with one byte per character
- Unicode allows multiple bytes per character
  - UTF-8 is the most common Unicode file encoding on Mac and Linux
  - Windows has its own encodings
- For ordinary letters and numbers, Unicode is identical to ASCII and to the encoding doesn’t matter

File connections

- When a piece of software accesses a file, it first **opens a connection** to the file
- Every connection has a **mode**:
  - Read, write, or append
  - Text or binary
- When the software is done with the file, it **closes** the connection
- You don't have to get to the end of the file to close the connection





Looping through a file in R

- If we get to the end of a file and run `readLines` again, it gives a character vector of length zero
- We can use the length of this vector as a test for whether the end of the file has been reached
- Use a `while` loop to keep reading and processing chunks until we get to the end of the file.

Mini exercise

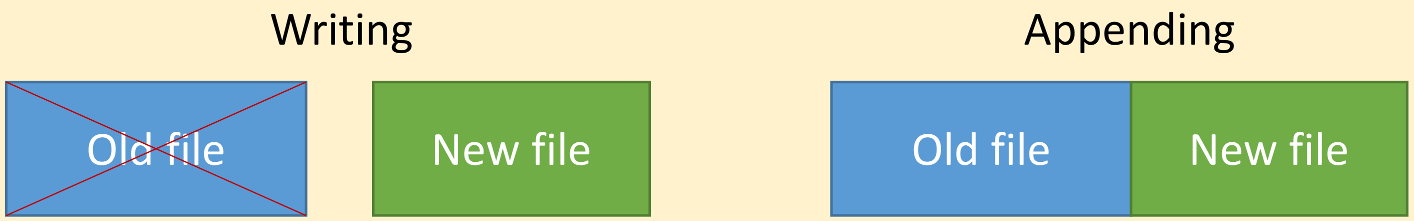
- Use `file` to open a reading connection to “Illumina_seq_example.fastq”
- Use `readLines` to preview the file a few lines at a time (some multiple of four)
- Use `file` to open a writing connection to a different text file
- Write just the sequences to the new file (second line out of every four) with `writeLines`
- Close both connections with `close`

File compression

- File is represented in a smaller number of bytes, at the expense of some processing time to compress or uncompress it
- Common compression formats
  - ZIP: mostly just on Windows. Allows multiple files.
  - GZIP: (GNU zip) open-source, very common on Mac and Linux
  - BZIP2: also open-source
  - See `?connections` for formats supported in R
- Compressed files can be accessed from beginning to end just like any other file, only with the added step that you compress or uncompress as you go.

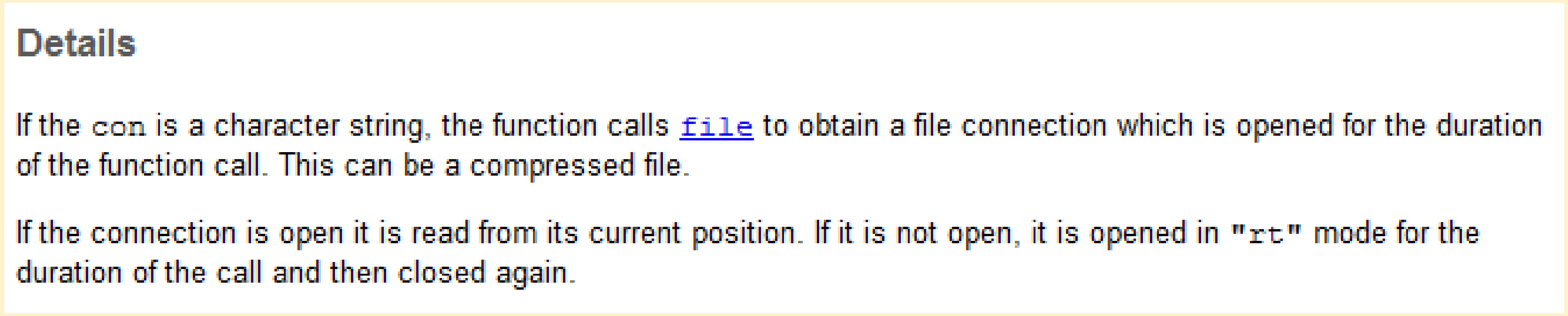
Writing vs. appending

- Do you want to overwrite a file, or write more data to the end of it?
- When you open a connection in mode ‘w’ (write), if the file exists it will be overwritten
- When you open a connection in mode ‘a’ (append), if the file exists you will add data to the end of it. If not you will start a new file.



Most R functions for reading and writing data will open and close the connections for you

From the `readLines` documentation:



Details

If the `con` is a character string, the function calls `file` to obtain a file connection which is opened for the duration of the function call. This can be a compressed file.

If the connection is open it is read from its current position. If it is not open, it is opened in `"rt"` mode for the duration of the call and then closed again.

- So basically, instead of passing `readLines` an open connection, you can pass it the file name if you want to read the file from the beginning.

So why bother learning to open and close connections?

- Useful if the file is too large to read into memory all at once
  - NGS sequence and alignment files can be many gigabytes
  - Read a piece of the file into RAM, use it for some computations, then discard it
- Useful if the file contains a mix of data types that you may want to read with different functions
  - Alignment and variant files have a header at the top that you might want to import with `readLines`, followed by tabular data you might want to import with `read.table`.

Reading delimited text

- Can follow a call to `readLines` with a call to `strsplit`

- For most tabular data, where columns are variables and rows are observations, we want to use the `read.table` family of functions
  - `read.table`: space delimiter (can set it to anything though)
  - `read.csv`: comma delimiter
  - `read.delim`: tab delimiter

Output of `read.table`

- An object of a class called **data frame**
- A data frame is a list of vectors where all vectors must be the same length, but not necessarily the same class
- Each vector (column) should usually represent some variable that was measured
- Each row (each element of each vector) should usually represent one observation

Subsetting a data frame

- Why did all the spaces get taken out of the column names? It is so they can be used as object names.
  - Any characters that are operators in R are also removed.
- We can use the dollar sign to extract one column
  - `mytable$Marker.name`
- We can get one column with double square brackets, like a list (useful if looping through columns)
  - `mytable[[1]]`
  - `mytable[[“Marker.name”]]`
- We can use single square brackets with a comma to specify row and column
  - `mytable[1:10, 1:4]`

`stringsAsFactors`

- This is an argument to all of the `read.table` functions that I almost always set to `FALSE` (default `TRUE`)
- If I am going to use a column as a categorical variable in linear regression, then it makes sense to have it as a factor
- For most bioinformatics applications, it is much better to keep a string as a string (gene names, DNA sequences, etc.)
- Sometimes it works to treat a factor as a string (e.g. `==`), but sometimes results are unexpected (e.g. `sort`, indexing)

Other `read.table` arguments

- `header`: does the file have column headers? (if not, first row is considered data)
- `sep`: what character(s) separate entries within a row?
  - Space: `" "`
  - Comma: `","`
  - Tab: `"\t“`
- `row.names`: 1 if the first column contains row names. Can also be a vector of row names themselves, or name of column with row names

Other `read.table` arguments

- `col.names`: column names, if not taken from the header
- `na.strings`: how to recognize missing values in character columns
  - Blanks are considered missing data in numeric but not character columns, by default
- `colClasses`: for specifying what class each column should be, in case R's best guess is incorrect
  - Example: experimental replications may be numbered, but you want to treat them as a factor
- `skip`: number of lines to skip before reading table (e.g. if the file has header lines in addition to col names)

Mini-exercise

- Make a call to `read.table` to correctly import "lecture5_germplasm_example.txt"
- Give `colClasses` a vector with the strings "character", "integer", and "Date" as appropriate
- Have it import "no data" as NA

`write.table`

- Take a data frame and export it to a file (or open connection)
- Many arguments similar to `read.table`
- `write.csv` ensures that the file can be opened in MS Excel

Differences between `read.table` and `write.table`

- First argument is data frame, second is file name
- To include column names in the file, use `col.names = TRUE` (not `header`)
- `row.names = TRUE` to include row names in output (always first column)
- Use `na` argument to indicate the string to print for missing data (not `na.strings`)
- `quote`: `TRUE` or `FALSE` to indicate whether strings should be in quotes

Mini-exercise

- Use `write.csv` to write the germplasm data frame that you imported out to a CSV
- Open it first with Notepad to see what it looks like
- Then open it in MS Excel
- Did Excel change anything?

Other functions for working with files

- For lots of detailed information, see “R Data Import/Export” on CRAN.
- `read.fwf`: if file is fixed-width rather than delimited
- `scan`: faster than `read.table` if you need to read lots of data that are all one class (less user-friendly)
- `cat`, `write`: For writing delimited text straight to file without column and row names
- `tempfile`: for making temporary files, automatically deleted when computer is shut down

Saving objects in your R environment

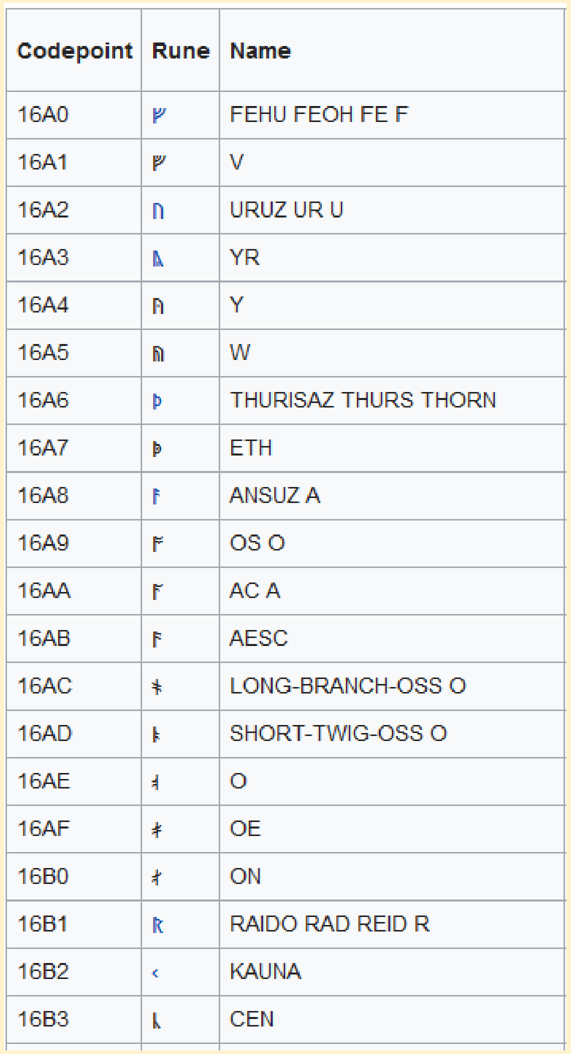
- `save.image` can write your whole R environment to an .RData file so you can reload it quickly later
- RStudio saves a file called ".RData" by default, but you can give files names like "germplasmWorkOct31.RData" etc.
- `save` is like `save.image` except you can specify which objects are saved (handy for large datasets)
- These files can't be opened by software other than R
- `load` is used to import these files into the R environment. Faster than reading data from text files.

(Extra material depending on time)

More on Unicode

- Includes
  - Characters from many languages worldwide
  - Characters from ancient languages
  - Symbols
  - Emojis
- More characters are being added every year
  - Codes for existing characters stay the same
- The first 128 characters are the same as ASCII
- If a character doesn’t display, it means you are using a font that doesn’t include that character

Codepoints in Unicode



Codepoint	Rune	Name
16A0	ᚠ	FEHU FEOH FE F
16A1	ᚡ	V
16A2	ᚢ	URUZ UR U
16A3	ᚣ	YR
16A4	ᚤ	Y
16A5	ᚥ	W
16A6	ᚦ	THURISAZ THURS THORN
16A7	ᚧ	ETH
16A8	ᚨ	ANSUZ A
16A9	ᚩ	OS O
16AA	ᚪ	AC A
16AB	ᚫ	AESC
16AC	ᚬ	LONG-BRANCH-OSS O
16AD	ᚭ	SHORT-TWIG-OSS O
16AE	ᚮ	O
16AF	ᚯ	OE
16B0	ᚰ	ON
16B1	ᚱ	RAIDO RAD REID R
16B2	ᚲ	KAUNA
16B3	ᚳ	CEN

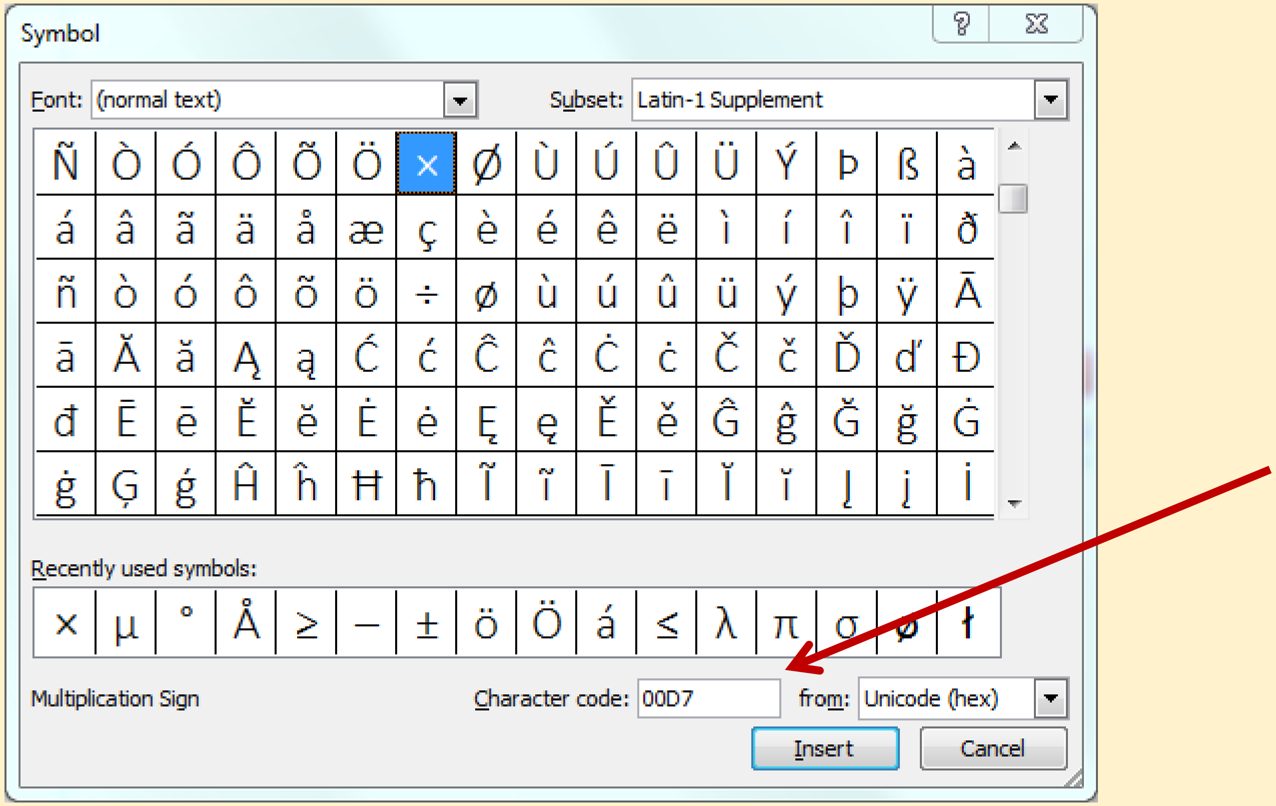
- If you look up Unicode characters, you'll often see four digits for the codepoint

- These are in hexadecimal
  - 0123456789ABCDEF
  - Decimal is 0123456789

- Two hex digits = one byte
  - $16^2 = 2^8$

Looking up codepoints

- “Insert symbol” dialogue in MS Office



Font: (normal text)

Subset: Latin-1 Supplement

Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß	à
á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï	ð
ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ	Ā
ā	Ă	ă	Ą	ą	Ć	ć	Ĉ	ĉ	Ċ	ċ	Č	č	Ď	ď	Đ
đ	Ē	ē	Ĕ	ĕ	Ė	ė	Ę	ę	Ě	ě	Ĝ	ĝ	Ğ	ğ	Ġ
ġ	Ģ	ģ	Ĥ	ĥ	Ħ	ħ	Ĩ	ĩ	Ī	ī	Ĭ	ĭ	Į	į	İ

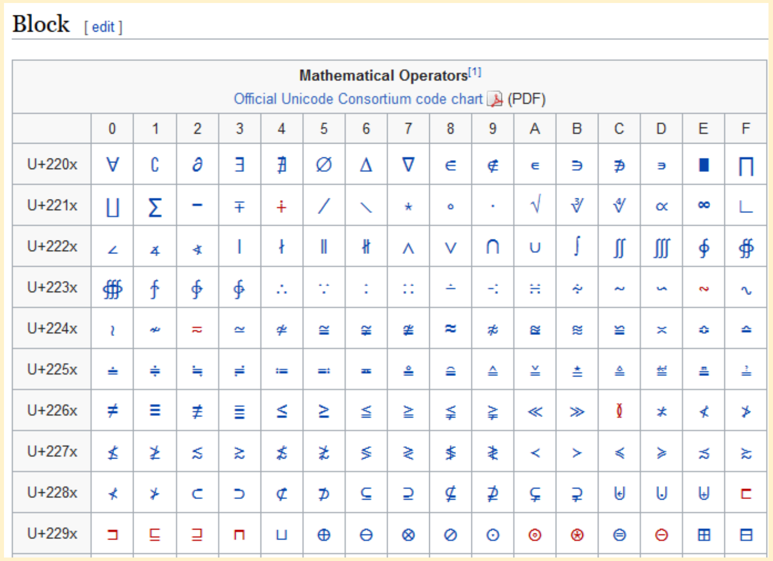
×	µ	°	Å	≥	–	±	ö	Ö	á	≤	λ	π	σ	ø	ł

Multiplication Sign

Character code: 00D7 from: Unicode (hex)

Looking up codepoints

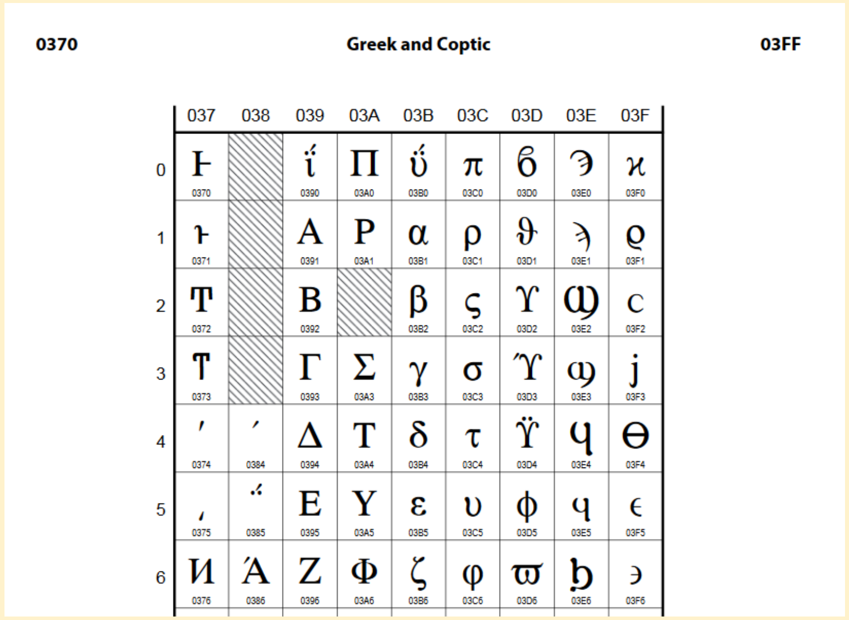
- Wikipedia has lots of Unicode tables
- www.unicode.org/charts has everything



Block [edit]

Mathematical Operators[1]
Official Unicode Consortium code chart (PDF)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+220x	∀	∁	∂	∃	∄	∅	∆	∇	∈	∉	∊	∋	∌	∍	∎	∏
U+221x	∐	∑	−	∓	∔	∕	∖	∗	∘	∙	√	∛	∜	∝	∞	∟
U+222x	∠	∡	∢	∣	∤	∥	∦	∧	∨	∩	∪	∫	∬	∭	∮	∯
U+223x	∰	∱	∲	∳	∴	∵	∶	∷	∸	∹	∺	∻	∼	∽	∾	∿
U+224x	≀	≁	≂	≃	≄	≅	≆	≇	≈	≉	≊	≋	≌	≍	≎	≏
U+225x	≐	≑	≒	≓	≔	≕	≖	≗	≘	≙	≚	≛	≜	≝	≞	≟
U+226x	≠	≡	≢	≣	≤	≥	≦	≧	≨	≩	≪	≫	≬	≭	≮	≯
U+227x	≰	≱	≲	≳	≴	≵	≶	≷	≸	≹	≺	≻	≼	≽	≾	≿
U+228x	⊀	⊁	⊂	⊃	⊄	⊅	⊆	⊇	⊈	⊉	⊊	⊋	⊌	⊍	⊎	⊏
U+229x	⊐	⊑	⊒	⊓	⊔	⊕	⊖	⊗	⊘	⊙	⊚	⊛	⊜	⊝	⊞	⊟



0370 Greek and Coptic 03FF

	037	038	039	03A	03B	03C	03D	03E	03F
0	Ͱ 0370	▨	ΐ 0390	Π 03A0	ΰ 03B0	π 03C0	ϐ 03D0	Ϡ 03E0	ϰ 03F0
1	ͱ 0371	▨	Α 0391	Ρ 03A1	α 03B1	ρ 03C1	ϑ 03D1	ϡ 03E1	ϱ 03F1
2	Ͳ 0372	▨	Β 0392	▨	β 03B2	ς 03C2	ϒ 03D2	Ϣ 03E2	ϲ 03F2
3	ͳ 0373	▨	Γ 0393	Σ 03A3	γ 03B3	σ 03C3	ϓ 03D3	ϣ 03E3	ϳ 03F3
4	ʹ 0374	΄ 0384	Δ 0394	Τ 03A4	δ 03B4	τ 03C4	ϔ 03D4	Ϥ 03E4	ϴ 03F4
5	͵ 0375	΅ 0385	Ε 0395	Υ 03A5	ε 03B5	υ 03C5	ϕ 03D5	ϥ 03E5	ϵ 03F5
6	Ͷ 0376	Ά 0386	Ζ 0396	Φ 03A6	ζ 03B6	φ 03C6	ϖ 03D6	Ϧ 03E6	϶ 03F6

Using Unicode in R

- If you are importing a file with special characters, you need to check the encoding to make sure they are read properly
- To manually put a character into a text string, type `\u` and then the four digits
- `"jalape\u00F1o"` = jalapeño
- Some characters have five digits and can't be used with the R base installation