Introduction to Mug!

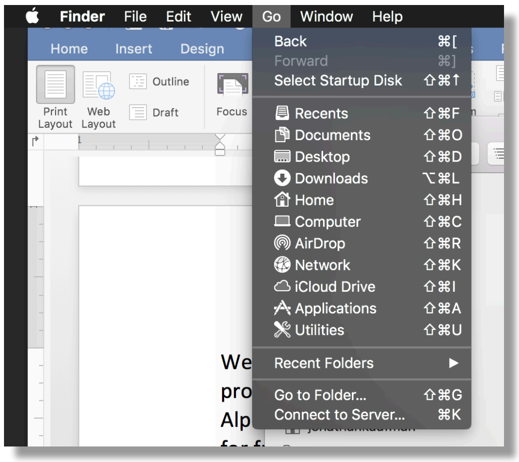
Awesome financial software for exploring equities.

By
Jonathan Kaufman, PhD

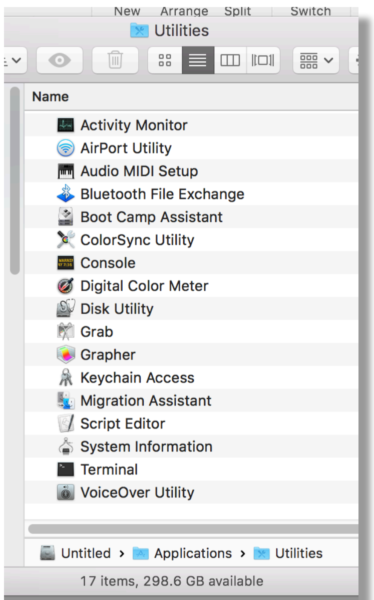
Getting Started on the Mac

Welcome to Mug. Awesome financial software for exploring equities. Mug currently provides smart interfaces to several financial data providors, including IEX Cloud, Alpaca, and Alphavantage.

Mug is currently provided as a command line application. Command line applications are accessed via a terminal window. To access a terminal window from a mac os x system, you can use the “Go” dropdown menu when having the system Finder in focus.



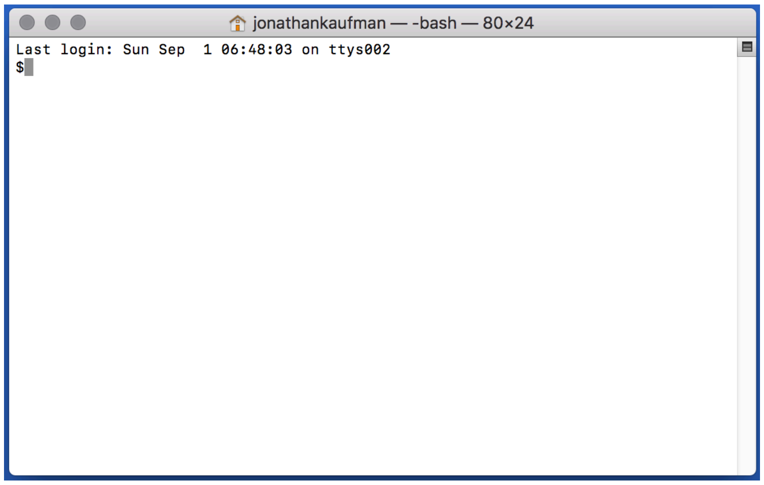
Then choose the “Utilities” menu item in the drop-down list. Note the utilities can also be opened with shift-command-U. When doing so, you should get a folder with similar contants as shown in figure TK.


Utilities

Name
Activity Monitor
AirPort Utility
Audio MIDI Setup
Bluetooth File Exchange
Boot Camp Assistant
ColorSync Utility
Console
Digital Color Meter
Disk Utility
Grab
Grapher
Keychain Access
Migration Assistant
Script Editor
System Information
Terminal
VoiceOver Utility

Untitled › Applications › Utilities
17 items, 298.6 GB available

Double-click the Terminal icon to open a terminal window. This terminal window will provide the command-line for the Mug! command line interface (CLI).



```
Last login: Sun Sep  1 06:48:03 on ttys002
$
```

First, however, we need to navigate our terminal window to the folder where the Mug application is located. This can be accomplished with the unix **cd** (cd for change directory) command. On my computer, the location is

```
      $ cd /Users/jonathankaufman/Dropbox/Mug
```

Note that the “**$**” is the unix prompt at the command line window. This prompt will vary depending on you computer’s settings. You should navigate to where you put the folder containing the Mug application as well as its associated data files. Then, once your terminal command line displays the correct folder location (which can be checked with the **pwd** (print working directory) command), you can (add your credentials such as secret keys for the services you will be using: iexcloud, alpaca, etc., and) initiate the Mug. Application with the following commend.

```
      $ java –jar Mug.jar
```

The use of the java command indicates that Mug currently runs on the Java Virtual Machine (JVM). If Java is not installed on you computer, the java command will of course not work. To install the latest version of Java on your computer, see the detailed instructions located at java.com.

To add your iexcloud credentials to Mug, open the file named keychain.clj in the resources subdirectory. The contents of the file look like as follows.

```
{:iexcloud        "pk_c38629bfd8mdfbddc62f3356"
 :alpaca1         "PKNAEQDJFSH6UXU1O0H"
 :alpaca2         "wx4n60KMWJYb0idmnvxdcmnHtf4NkQUU8H31lE/vK"
 :alphavantage    "8T1KDJSDNVDYCWK"}
```

It is readily apparent where to put your secret keys and other credentials so that they can be used by Mug. The Mug software will use this file to access your accounts on the listed financial data providors.

Once Java and Mug are correctly installed, and your credentials entered into the `keychain.clj` file, the following prompt should appear when you first launch the Mug program using the java command indicated above **(**`$ java -jar mug.jar`).

```
Welcome to Mug!
******************

(For help, enter .h (including the dot) at the prompt >)

top>
```

The "top>" character is the top-level prompt where you provide instructions to Mug!. Most commands at this prompt begin with the period symbol "**.**" For example, if you want to see a list of companies within a specific market cap range (or window), use the *window* command which is simply **`.w`** followed by the *low* and *high* limits of the range (in millions USD) . So, for example, the command

```
top> .w 100 200
```

provides a list of companies within the market cap range from $100 million to $200 million. When doing so, the Mug system change state. This is evident by the new prompt which looks like

```
:fresh>
```

The colon indicates that we are no longer in the top-level state of Mug!, but are alternatively in a collection. The word **`fresh`** in the prompt indicates that the name of the collection that is loaded is "fresh." This is the default name. You can create a name for any collection that is in focus using the **`.s`** (save) command.

```
:fresh> .s green
:green>
```

Here we see that the **`.s green`** *command* named the collection "green," and the temporary name, "fresh," remains available for the generation of new collection. Use **`.l`** (l for list) to get a list items in a collection (the same **`.l`** command provides a list of collection names when used at the top level), and use **`.h`** (h for help) to get a list of additional commands.

If a *command* is not preceeded by a "**.**" Mug! interprets the text to be either a symbol name or the name of a named collection – and brings such symbol or collection into focus. Assuming we typed in the symbol `mrk` at the prompt, the prompt updates and now starts with a "." Rather than a ":". The "." At the beginning of the prompt indicates that the Mug! focus is now on a single company rather than a collection of companies (which would be indicated by the promt starting with a ":". And also the symbol for this single company is displayed in the prompt so that the Mug! user can be aware of state.

```
top>     .w 100 200
:fresh> .s green
:green> mrk
.MRK>
```

When Mug has a single company in focus, additional "**.**"-preceeded commands are available that provide data on the company in focus. For example, to find the cash position, use **.c**, and for the market cap, use **.mkt**. The complete list of such commands is shown when typing **.h** at the prompt when a single company is in focus (which as mentioned is indicated by the prompt beginning with a "**.**" rather than "**:**").

Making Tables

Collections can be transformed into tables using the **.m** (m for map) command when a collection is in focus. Let's say we have a collection named fresh and it is in focus. (A simple way to obtain such state is to give the **.w** (w for window) command at the top level). You can obtain a table of say each company's market cap and cash position as follows.

```
:fresh> .m c mkt
t      c      mkt
CRNX   163    634
AKBA   321    714
LXRX   160    747
KURA   178    590
SRNE   168    614
CTST   52     556
VKTX   301    600
```

Here we have "mapped" (using **.m** command) the functions **c** for cash and **mkt** for market cap. Notice that unlike the **.m** command, the **c** and **mkt** functions are not preceeded with a "**.**". We can have any number of functions we like. The following table simply adds volume data.

```
:fresh> .m c mkt v
t      c      mkt    v
CRNX   163    634    0.88
AKBA   321    714    5.06
LXRX   160    747    0.89
KURA   178    590    2.47
SRNE   168    614    4.11
CTST   52     556    6.51
VKTX   301    600    26.56
```

Notice how potential outliers can be obtained with such tables, witness the volume for VKTX. We can explore if that may be related to revenue as follows.

```
:fresh> .m c mkt v r
t     c     mkt   v     r
CRNX  163   634   0.88  0
AKBA  321   714   5.06  207
LXRX  160   747   0.89  63
KURA  178   590   2.47  0
SRNE  168   614   4.11  21
CTST  52    556   6.51  35
VKTX  301   600   26.56 0
```

Here we discover no link to revenue between revenue and volume in this collection. We may be interested in mapping arithmatic combinations of these functions. For example, the cash to market cap ratio. This goal can be accomplished by putting the desired arithmatic operator (**+**, **–**, *,or **!**) between the two functions. Notice importantly that we use the symbol **!** for division rather than a slash. Here's how simple is it to create the table.

```
:fresh> .m c mkt v c!mkt
t     c     mkt   v     c!mkt
CRNX  163   634   0.88  0.25
AKBA  321   714   5.06  0.44
LXRX  160   747   0.89  0.21
KURA  178   590   2.47  0.3
SRNE  168   614   4.11  0.27
CTST  52    556   6.51  0.09
VKTX  301   600   26.56 0.5
```

Sometimes the cash over market cap **c!mkt** values significantly exceed 1. This is usually only for microcap companies and/or companies that have completely lost the value of other assets (say with the failure of a clinical trial). Here, in this collection, the highest is 0.5 for VKTX. We can get additional information on a single security when a collection is in focus by simply including the functions after the security name when responding to the collection prompt. For example, suppose we want a list of institutional ownership for VKTX. We can use the **io** command.

```
:fresh> vktx io
2019 Vanguard_Group_Inc__Subfiler_
2019 State_Street_Corp
2019 Sio_Capital_Management_LLC
2019 Park_West_Asset_Management_LLC
2019 Northern_Trust_Corp
2019 Fiera_Capital_Corp
2019 Fidelity_Management_&_Research_Co
2019 BlackRock_Institutional_Trust_Co_NA
2019 BlackRock_Fund_Advisors
2019 ArrowMark_Colorado_Holdings_LLC
```

Or perhaps we are interested in financing cash flow (**cffd** cash flow financing detail).

```
:fresh> vktx cffd
20190630 0.26
20190331 0.32
```

```
20181231 0.34
20180930 167.48
20180630 70.14
20180331 62.29
20171231 14.76
20170930 1.11
20170630 4.0
20170331 2.52
20161231 1.03
20160930 0.1
```

Here we see a history of financing cash flow by quarter, the largest such cash flow ocurring in the quarter ending September 30, 2018. Suppose you want a function that provides an indication of the extent of financing cash flow to be used in the **`.m`** function when creating a table. For this purpose, you can use the scalar-value **`cff`** instead of **`cffd`**, as follows.

```
:fresh> .m c mkt cff
t     c     mkt   cff
CRNX  163   634   170
AKBA  321   714   212
LXRX  160   747   147
KURA  178   590   296
SRNE  168   614   436
CTST  52    556   127
VKTX  301   600   324
```

Here, the **`cff`** function *reduces* the result of **`cffd`** so that its result can be used as a single number, which is a requirement for creating a simple table and thus a requirement for use in the **`.m`** command at a collection prompt. In this case, the sum of all the reported quarterly cash flows is provided. A list of such functions can be obtained at the command prompt by typing the composite help command **`.h cmd`**.

Exploring Symbols

Suppose you want to focus on a single company for exploration of various functions or data that cannot be succinctly represented on maps. Then by all means, as mentioned above at the end of the “Getting Started” section, simply enter the symbol for the company without including any function in the command. Let’s do that for `AKBA`.

```
:fresh> akba
.AKBA>
```

Notice that the prompt subsequently begins with a “**`.`**” and not a “**`:`**”. This is the indicator that we have a single company in focus rather than a named collection of companies. Now that we have a single company in focus, we can use all of the company-focused functions directly by treating them as *commands*, and preceeding them with the “**`.`**” symbol.

Let’s try simply showing the cash position. Now be careful to use **.c** instead of just **c** . If you leave out the **.** before the **c**, Mug will think you want to look at a company that has the symbol **C**, which currently happens to be Citigroup.

```
.AKBA> .c
321
.AKBA> c
.C> .cname
Citigroup
```

As we would expect, returning to an existing *named group* (or the most recent version of the automatically generated **fresh** *group*) is as easy as typing the name of the *group* at the prompt.

```
.C> fresh
:fresh>
```

You can also return to the most recent group that was in focus with the **.u** (u for up) *command*. This *command* will also return Mug!’s focus to **top>**, if that was the focus prior to the *company* focus. Using the **.u** command when the focus is on a collection will also bring Mug!’s focus to **top>**.

Yahoo, Edgar, and Company Websites

One of Mug’a design principles is to not reinvent the wheel. If there is a free service available that offers dynamic price history charts such as yahoo finance, why re-invent that? Mug will simply open a yahoo chart window for you with the **.yahoo** command. This command can be used at a symbol-focused prompt, or it can be used without the preceeding **.** if it is preceeded by the symbol name at the **top>** and/or **:collection>** *level* prompts.

Similarly, Mug! will open the relevant `www.sec.gov` website page in response to the **.sec** command. Regarding both of these convenience *commands*, in practice, they are much faster and easier to use than the alternative of manually navigating a browser window/tab to the desired page. These as well as the **.oweb** command that opens a browser window to the desired company’s website where additional basic information can be found. These commands are often the fastest way navigate to the exact websites of interest.

Updating Sets of Companies

Earlier in this introduction we created a set of companies using the **.w** (w for window) command. Recall that **.w** took two numbers as the *low* and *high* values of market cap to include in the set. Essentially .w 50 60, generates the set of companies having market caps within the range of $50M to $60M. Let’s experiment with a narrow range so the we can look at just a few companies.

```
top> .w 60 65
:fresh> .l
OVID  Ovid Therapeutics
HTGM  HTG Molecular Diagnostics
SCPH  scPharmaceuticals
GNCA  Genocea Biosciences
ALIM  Alimera Sciences
INSY  INSYS Therapeutics
EYEN  Eyenovia
AQXP  Aquinox Pharmaceuticals
:fresh>
```

suppose we want to add a company to the set. We can do this by adding the company via its ticker symbol and the **.a** (a of add) command as follows.

```
:fresh> .a mrk
:fresh> .a pfe
:fresh> .l
OVID  Ovid Therapeutics
HTGM  HTG Molecular Diagnostics
SCPH  scPharmaceuticals
GNCA  Genocea Biosciences
ALIM  Alimera Sciences
INSY  INSYS Therapeutics
EYEN  Eyenovia
AQXP  Aquinox Pharmaceuticals
MRK   Merck & Co.
PFE   Pfizer Inc.
:fresh>
```

We can notice apparent differences between these companys by using the **.m** (m for map) command as earlier described.

```
:fresh> .m c r e
t     c     r     e
OVID  41    0     -52
HTGM  31    21    -15
SCPH  89    0     -29
GNCA  26    0     -38
ALIM  13    46    -7
INSY  95    82    -101
EYEN  19    0     -17
AQXP  76    0     0
MRK   8864  42446 14292
PFE   18833 53647 21673
:fresh>
```

The table above is a listing of cash (**c**), revenue (**r**), and earnings (**e**). Notice how Merck (MRK) and Pfizer (PFE) are so much larger than the other companies (obtained with the **.w** function and a $65M upper limiy on market cap. This is one method for finding outliers of interest in our data tables. There may be column-specific numbers that are inconsistent with those of the members of the set.

A new set can be constructed from scratch (rather than using the **`.w`** (w for window) command). To do so, use the **`.b`** (b for bag) command. The **`.b`** command works by simply adding the specified stock-tickers (representative of the to-be-added companies)

```
top> .b mrk pfe msft ibm ba
:fresh> .l
MRK   Merck & Co.
PFE   Pfizer Inc.
MSFT  Microsoft Corp.
IBM   International Business Machines Corp.
BA    The Boeing Co.
:fresh>
```

The syntax of this command is simplt **`.b`** followed by the tickers of interest. In addition to adding tickers to the set (as shown earlier), tickers can similarly be removed from the set with the **`.d`** (d for delete) command. Currently the .d command only takes a single argumant, but it will soon be expanded to take any number of arguments. The following listing is an example of msft being removed from the *fresh* list.

```
:fresh> .l
MRK   Merck & Co.
PFE   Pfizer Inc.
MSFT  Microsoft Corp.
IBM   International Business Machines Corp.
BA    The Boeing Co.
:fresh> .d msft
:fresh> .l
MRK   Merck & Co.
PFE   Pfizer Inc.
IBM   International Business Machines Corp.
BA    The Boeing Co.
:fresh>
```

Notice in the listing above, MSFT was successfully removed from the list using the **`.d`** command.