

---

# **Avendesora Documentation**

*Release 1.10.5*

**Ken Kundert**

**Mar 31, 2018**



---

## Contents

---

<b>1</b>	<b>What is Avendesora?</b>	<b>3</b>
<b>2</b>	<b>Quick Tour</b>	<b>5</b>
<b>3</b>	<b>Contents</b>	<b>9</b>
<b>4</b>	<b>Indices and tables</b>	<b>83</b>



Version: 1.10.5

Released: 2018-03-28

Please report all bugs and suggestions at [Github](#) (or contact me directly at [avendesora@nurdletech.com](mailto:avendesora@nurdletech.com)).



---

## What is Avendesora?

---

*Avendesora* holds all of your online account information. In addition to the username and password, it holds any account information you might want such as account numbers, PINs, verbal passwords, one-time passwords, security questions, URLs, email addresses, phone numbers, etc. *Avendesora* is a secure repository for all of this information, using GPG to keep the information safe.

Account secrets, such as passwords and such can either be saved, as with password vaults, or they can be generated by *Avendesora*. Generation is quite flexible and is generally preferred as it makes the secrets extremely hard to predict, in most cases eliminating the risk they could be cracked. *Avendesora* generates secrets from a random seed. The seed can be shared with a collaborator, and once shared, either collaborator can create new shared passwords.

You can query *Avendesora* directly from the command line. When doing so you can either display account information or copy it to the clipboard. You can also configure a hot-key to run *Avendesora*, in which case it determines which information is needed from context and then fills it into the active application. In this way *Avendesora* can directly enter account information into your browser, email client, shell, etc. The information *Avendesora* provides can be used to log you in, answer security questions, enter your credit card number, etc.

*Avendesora* is a program that is deeply steeped in Unix traditions. It operates primarily from the command line and leans heavily on programs you are likely already familiar with, such as Python, GPG, and Vim. As such, it should be both welcoming and powerful for those that are comfortable with Unix and its utilities. Also, it is fully open source, so you can change it if you do not like some aspect of it. Please consider contributing your enhancements back to the project. Here are some of the ways *Avendesora* differs from other available password managers:

1. Completely offline
2. Linux only
3. Command line centric
4. Completely free (free as in freedom and free as in beer)
5. Supports collaboration
6. Supports time-based one-time passwords (ala Google Authenticator)
7. Uses GPG for encryption
8. Written in Python

9. Provides a Python API



## CHAPTER 2

---

### Quick Tour

---

With *Avendesora* you create files that contain information about your accounts. *Avendesora* accesses that information and shows it to you when you need it. The files can be encrypted with GPG, and so are quite secure. The information itself is grouped into accounts, with an account consisting of both secret and non-secret information. The non-secret information includes such things as user names, email addresses, phone numbers, etc. The secret information includes passwords, pins, security questions and such. Information is free form. You decide what information you want to associate with an account, what you call it, and whether it is secret or not. There are two types of secrets: remembered secrets and generated secrets.

In general, it is best to use generated secrets if you can. They are preferred for two reasons. First, generated passwords are pretty much assured of having high entropy, and entropy in your passwords is like fiber in your diet, the more the better because it results in passwords that resist cracking. Second, you can easily share generated secrets with your collaborators without risk of exposing to secrets to others.

As a demonstration, consider adding an account for FasTrak, a payment service for toll roads in the San Francisco bay area. First you would add the account:

```
> avendesora add website
```

This indicates that *Avendesora* should create a new account in the default accounts file based on the *website* template.

*Avendesora* responds by opening your editor with a rough template containing the fields needed for a typical website account. You should modify it to suit your needs. For example, your entry for FasTrak might look like this:

```
class FasTrak(Account):
    desc = 'payment service for automated toll collection'
    aliases = 'fastrack fasttrack'
    username = 'orchid3649'
    email = 'orchid3649@yahoo.com'
    passcode = PasswordRecipe('12 2u 2d 2s')
    discovery = RecognizeURL(
        'https://www.bayareafastrak.org',
        script=' {username} {tab} {passcode} {return}'
    )
    questions = [
        Question('What city were you born in?')
```

```
    Question('What was the name of you high school?')
]
pin = PIN(length=4)
```

This is Python code. An account is created by declaring a subclass of Account. The account information is given as class attributes. *Avendesora* supports string, list, and dictionary attributes. You create secrets by instantiating a Secrets class. This example uses three different secrets, all of which are generated: PasswordRecipe(), Question() and PIN(). First consider PIN(). Notice that you do not give a PIN number, you instead just specify how long it should be. *Avendesora* generates a PIN for you at random. With PasswordRecipe() you do not specify the password, you specify how long it should be and what kind of characters it should use (in this case, 12 long including 2 uppercase, 2 digits, and 2 symbols). Question() is used to generate random answers to security questions. Again, you do not give the answer, you give the question and the answer is generated at random. It is the unpredictability of these values that make them secure.

Once the information is entered for your account, you can see the values by running the following commands (of course if you try this example your results will differ):

```
> avendesora value fastrak passcode
passcode: 0GPD;mc3XC?c

> avendesora value fastrak questions.0
questions.0 (What city were you born in?): voyager interview gaudy

> avendesora value fastrak pin
pin: 2728
```

You can also access the account values that are not secret in a similar manner:

```
> avendesora value fastrak username
username: orchid3649
```

The difference is that *Avendesora* erases secrets from the screen after displaying them for a minute, which is not done with non-secrets.

There are various tricks available to reduce the amount you type. For example:

```
> avendesora fastrak
username: orchid3649
passcode: 0GPD;mc3XC?c
```

If you give an account name without a command, the *credentials command* is run, which displays the username and password for the specified account.

```
> avendesora fastrak pin
pin: 2728
```

In this case the account and field name was given, but not a command name. When more than one argument is given, and the first is not recognized as a command, the *value command* is run.

The *discovery* attribute is used by *Avendesora* to associate an account to a URL or URLs. You can visit the FasTrak website using:

```
> avendesora b fastrak
```

The *b* is short for *browse* (the most common *Avendesora* commands have single character aliases). Running that command opens your browser if it is not already open, and navigates to the FasTrak URL. Generally you would run this command directly from your window manager, which allows you to navigate to your account without opening a shell.

The information provided to *discovery* also allows the desired account to be recognized, which allows you to directly enter values into an application, in this case the web browser, with a single keystroke. To do so, you would associate *Avendesora* with a keyboard shortcut (a hot key), such as Alt-a ('a' for Avendesora) or Alt-p ('p' for password). Once the webpage is open, simply click on the *Username* field and type your shortcut (Alt-p). This runs *Avendesora*, which then looks at the current environment to determine which account to use. In the case of `RecognizeURL()` it is looking for the URL in the browser's window title. *Avendesora* checks with all the accounts and finds that only FasTrak matches, at which point it executes the given script, which produces the user name and passcode.

This approach is a very secure way to access your account because:

1. Using the *browse command* assures you are using a known-good URL, preventing you from being phished.
2. If you do fall prey to a phishing scheme, *Avendesora* will not recognize the URL and so will not disclose your account credentials.
3. *Avendesora* warns you if you are attempting to provide your account credentials to an insecure webpage (an http page rather than an https page).

Here are some other convenient *Avendesora* commands.

The *edit command* opens an account in your editor, allowing you to update the account values:

```
> avendesora edit amtrak
```

The *find command* finds accounts whose name contain a string of characters in the name or alias. Notice that I tend to add common misspellings as aliases.

```
> avendesora find track
track:
  amtrak (amtrak)
  fastrak (fastrack, fasttrack)
  python-bug-tracker
```

The *search command* finds accounts whose attributes contain a string of characters. Only attributes whose values are not secret are examined.

```
> avendesora search junior
junior:
  gmail
  fidelity
```

The *values command* prints out a summary of all the account attributes. The secrets are not printed with this command.

```
> avendesora values fastrak
names: fastrak, fastrack, fasttrack
email: orchid3649@yahoo.com
passcode: <reveal with 'avendesora value fastrak passcode'>
pin: <reveal with 'avendesora value fastrak pin'>
questions:
  0: What city were you born in? <reveal with 'avendesora value fastrak questions.0
  ↵>
  1: What was the name of you high school? <reveal with 'avendesora value fastrak_
  ↵questions.1'>
username: orchid3649
```

Finally, you can use the *help command* to get information on the various commands and other useful topics.



## 3.1 Conceptual Underpinnings

### 3.1.1 Generated Secrets

Account secrets can be saved in encrypted form, as with password vaults, or generated from a root secret. Generated secrets have several important benefits. First, they are produced from a random seed, and so are quite unpredictable. This is important, because the predictability can be exploited when cracking passwords. Second, if the root secret is shared with another trusted party, then you both can generate new shared secrets without passing any further secrets. Furthermore, if you keep a copy of the root secret, say in a safe deposit box, then there is a good chance you can resurrect your secrets if you happen to lose your accounts files. Finally, with generated secrets, it is possible to have *stealth secrets*, which are secrets for which there is absolutely no evidence.

Secrets are generated from a collection of seeds, one of which must be random with a very high degree of entropy. The random seed is referred to as the ‘master seed’. It is extremely important that the master seed remain completely secure. Never disclose a master seed to anyone except for a person you wish to collaborate with, and then only use the shared master seed for shared secrets. Each file that contains accounts will contain a master seed for the accounts it holds. Typically, you would have one file to hold your private accounts, and then one file for every group of people you collaborate with.

A secret is generated by combining a master seed with several other seeds, such as the account name, the secret name, and perhaps a version name. The combination is then hashed to form a long binary number that is unique to your secret. From there the number is transformed into a usable form by one of the Secrets classes. `PIN()` convert it to a sequence of digits, `Password()` converts it to a sequence of characters, `Passphrase` converts it to a sequence of words, etc.

For example, consider the following rather abbreviated accounts file:

```
from avendesora import Account, Passphrase

master_seed = 'c2VjcmV0IG1lc3NhZ2UsIHN1Y2Nlc3NmdWxseSBkZWVvZGVkIQ'

class Login(Account):
```

```
username = 'herbie'  
passcode = Passphrase()
```

This file contains one secret, the login passphrase for Herbie. In this case, the master seed is combined with the words 'login' (the account name, down cased) and 'passcode' (the attribute name); the combination is hashed, and the hash is used to generate the passphrase. The words in the passphrase are chosen at random from a dictionary of roughly 10,000 words. The first word is chosen by taking the first 14 bits from the hash and using that to number to select a word. The second word is chosen using the next 14 bits, and so on. The hash is constructed such that even the smallest changes in any seed results in a completely different hash. As such, the resulting passphrase is quite unpredictable.

The passcode itself is not stored, rather it is the seeds that are stored and the passcode is regenerated when needed. Notice that all the seeds except the master seed need not be kept secure. Thus, once you have shared a master seed with a collaborator, all you need to do is share the remaining seeds and your collaborator can generate exactly the same passcode.

Another important thing to notice is that the generated passcode is dependent on the account and secret names. Thus if you rename your account or your secret, the passcode will change. So you should be careful when you first create your account to name it appropriately so you don't feel the need to change it in the future.

### 3.1.2 Entropy

A 4 word Avendesora password provides 53 bits of entropy, which seems like a lot, but NIST is recommending 80 bits for your most secure passwords. So, how much is actually required? It is worth exploring this question.

Entropy is a measure of how hard the password is to guess. Specifically, it is the base two logarithm of the likelihood of guessing the password in a single guess. Every increase by one in the entropy represents a doubling in the difficulty of guessing your password. The actual entropy is hard to pin down, so generally we talk about the minimum entropy, which is the likelihood of an adversary guessing the password if he or she knows everything about the scheme used to generate the password but does not know the password itself. So in this case the minimum entropy is the likelihood of guessing the password if it is known that we are using 4 space separated words as our passphrase where the words are selected at random with a uniform distribution from a known list. This is very easy to compute. There are roughly 10,000 words in our dictionary, so if there was only one word in our passphrase, the chance of guessing it would be one in 10,000 or 13 bits of entropy. If we used a two word passphrase the chance of guessing it in a single guess is one in  $10,000 * 10,000$  or one in 100,000,000 or 26 bits of entropy.

The probability of guessing our passphrase in one guess is not our primary concern. Really what we need to worry about is given a determined attack, how long would it take to guess the password. To calculate that, we need to know how fast our adversary could try guesses. If they are trying guesses by typing them in by hand, their rate is so low, say one every 10 seconds, that even a one word passphrase may be enough to deter them. This is why bank PINs can be so short. Our one word passphrase provides roughly the same security as a four digit PIN. Alternatively, they may have a script that automatically tries passphrases through a login interface. Again, generally the rate is relatively slow. Perhaps at most they can get is 1000 tries per second. In this case they would be able to guess a one word passphrase in 10 seconds and a two word passphrase in a day, but a 4 word passphrase would require 300,000 years to guess in this way.

The next important thing to think about is how your password is stored by the machine or service you are logging into. The worst case situation is if they save the passwords in plain text. In this case if someone were able to break in to the machine or service, they could steal the passwords. Saving passwords in plain text is an extremely poor practice that was surprisingly common, but is becoming less common as companies start to realize their liability when their password files get stolen. Instead, they are moving to saving passwords as hashes. A hash is a transformation that is very difficult to reverse, meaning that if you have the password it is easy to compute its hash, but given the hash it is extremely difficult to compute the original password. Thus, they save the hashes (the transformed passwords) rather than the passwords. When you log in and provide your password, it is transformed with the hash and the result is compared against the saved hash. If they are the same, you are allowed in. In that way, your password is not stored and so is no longer available to thieves that break in. However, they can still steal the file of hashed passwords, which

is not as good as getting the plain text passwords, but it is still valuable because it allows thieves to greatly increase the rate that they can try passwords. If a poor hash was used to hash the passwords, then passwords can be tried at a very high rate. For example, it was recently reported that password crackers were able to try 8 billion passwords per second when passwords were hashed with the MD5 algorithm. This would allow a 4 word passphrase to be broken in 14 days, whereas a 6 word password would still require 4,000,000 years to break. The rate for the more computational intensive sha512 hash was only 2,000 passwords per second. In this case, a 4 word passphrase would require 160,000 years to break.

In most cases you have no control over how your passwords are stored on the machines or services that you log into. Your best defense against the notoriously poor security practices of most sites is to always use a unique password for sites where you are not in control of the secrets. That way the poor security practices of one site would not compromise your other accounts. For example, you might consider using the same passphrase for you login password and the passphrase for an ssh key on a machine that you administer, but never use the same password for two different websites unless you do not care if the content of those sites become public.

So, if we return to the question of how much entropy is enough, you can say that for important passwords where you are in control of the password database and it is extremely unlikely to get stolen, then four randomly chosen words from a reasonably large dictionary is plenty. If what the passphrase is trying to protect is very valuable and you do not control the password database (ex., your brokerage account) you might want to follow the NIST recommendation and use 6 words to get 80 bits of entropy. If you are typing passwords on your work machine, many of which employ keyloggers to record your every keystroke, then no amount of entropy will protect you from anyone that has or gains access to the output of the keylogger. In this case, you should consider things like one-time passwords or two-factor authentication. Or better yet, only access sensitive accounts from your home machine and not from any machine that you do not control.

## 3.2 Installing and First Use

Install with:

```
pip3 install --user avendesora
```

This will place avendesora in `~/local/bin`, which should be added to your path.

You will also need to install some operating system commands. On Redhat systems (Fedora, Centos, Redhat) use:

```
yum install gnupg2 xdotool xsel
```

You should also install `python-gobject`. Conceivably this could be installed with the above `pip` command, but `gobject` appears broken in `pypi`, so it is better use the operating system's package manager to install it. See the `setup.py` file for more information. On Redhat systems use:

```
yum install python3-gobject
```

`Gobject` is only used to provide a user-interactive selection utility. However, if you prefer, you can use `dmenu` for your selection utility, in which case you will need to install it by hand using:

```
yum install dmenu
```

If you would like to use `scrypt` as a way of encrypting fields, you will need to install `scrypt` by hand using:

```
pip3 install --user scrypt
```

### 3.2.1 GPG Key

To use *Avendesora*, you will need GPG and you will need a GPG ID that is associated with a private key. That GPG ID could be in the form of an email address or an ID string that can be found using ‘`gpg --list-keys`’.

If you do not yet have a GPG key, you can get one using:

```
$ gpg --gen-key
```

You should probably choose 4096 RSA keys. Now, edit `~/.gnupg/gpg-conf` and add the line:

```
use-agent
```

That way, you generally need to give your GPG key passphrase less often. The agent remembers the passphrase for you for a time. Ten minutes is the default, but you can configure `gpg-agent` to cache passphrases for as long as you like.

If you use the agent, be sure to also use screen locking so your passwords are secure when you walk away from your computer.

### 3.2.2 Vim

If you use Vim, it is very helpful for you to install GPG support in Vim. To do so first download:

```
http://www.vim.org/scripts/script.php?script\_id=3645
```

Then copy the file into your Vim configuration hierarchy:

```
cp gnupg.vim ~/.vim/plugin
```

### 3.2.3 Initializing Avendesora

To operate, *Avendesora* needs a collection of configuration and accounts files that are stored in `~/.config/avendesora`. To create this directory and the initial versions of these files, run:

```
avendesora init -g <gpg_id>
```

For example:

```
avendesora init -g rand@dragon.com
```

or:

```
avendesora init -g 1B2AFA1C
```

If you would like to have more than one person access your passwords, you should give GPG IDs for everyone:

```
avendesora init -g rand@dragon.com,lews.therin@dragon.com
```

After initialization, there should be several files in `~/.config/avendesora`. In particular, you should see at least an initial accounts files and a config file.



### 3.2.4 Initial Configuration

The config file (`~/.config/avendesora/config`) allows you to personalize *Avendesora* to your needs. The available configuration settings are documented in `~/.config/avendesora/config.doc`. After initializing your account you should take the time to review your configuration and adjust it to fit your needs. You should be very thoughtful in this initial configuration, because some decisions (or non-decisions) you make can be very difficult to change later. The reason for this is that they may affect the passwords you generate, and if you change them you may change existing generated passwords. In particular, be careful with *dictionary\_file*. Changing this value when first initializing *Avendesora* is fine, but should not be done or done very carefully once you start creating accounts and secrets.

During an initial configuration is also a convenient time to determine which of your files should be encrypted with GPG. To assure that a file is encrypted, give it a GPG file suffix (`.gpg` or `.asc`). The appropriate settings to adjust are: *archive\_file*, *log\_file*, both of which are set in the config file, and the accounts files, which are found in `~/.config/avendesora/accounts_files`. For security reasons it is highly recommended that the archive file be encrypted, and any accounts file that contain sensitive accounts. If you change the suffix on an accounts file and you have not yet placed any accounts in that file, you can simply delete the existing file and then regenerate it using:

```
avendesora init -g <gpg_id>
```

Any files that already exist will not be touched, but any missing files will be recreated, and this time they will be encrypted or not based on the extensions you gave.

More information on the various configuration options can be found in [Configuring](#).

### 3.2.5 Configuring Your Window Manager

You will want to configure your window manager to run *Avendesora* when you type a special hot key, such as `Alt p`. The idea is that when you are in a situation where you need a secret, such as visiting your bank's website in your browser, you can click on the username field with your mouse and type your hot key. This runs *Avendesora* without an account name. In this case, *Avendesora* uses *account discovery* to determine which secret to use and the script that should be used to produce the required information. Generally the script would be to enter the username or email, then tab, then the passcode, and finally return, but you can configure the script as you choose. This is all done as part of configuring discovery. The method for associating *Avendesora* to a particular hot key is dependent on your window manager.

Gnome:

With Gnome, you must open your Keyboard Shortcuts preferences and create a new shortcut. When you do this, choose 'avendesora value' as the command to run.

I3:

Add the following to your I3 config file (`~/.config/i3/config`):

```
bindsym $mod+p exec --no-startup-id avendesora value
```

OpenBox:

Key bindings are found in the `<keyboard>` section of your `rc.xml` configuration file. Add a key binding for *Avendesora* like this:

```
<keyboard>
...
  <keybind key="A-p">
    <action name="Execute">
      <command>avendesora value</command>
    </action>
  </keybind>
```

```
...  
</keyboard>
```

## 3.2.6 Configuring Your Browser

Finally, to improve account discovery, it is recommended that you add a plugin to your web browser that puts the URL into the window title. How to do so is described in *Account Discovery*.

## 3.3 Overview

Use of *Avendesora* will be illustrated through a series of examples. However, before starting it is helpful to know that *Avendesora* provides several commands to help you use it. First, it provides a *help command*:

```
> avendesora help
```

This lists the available help topics. You can ask about a specific topic using:

```
> avendesora help <topic>
```

Adding the `-browse` option allows you to access the online version of the manual through your web browser. For example,

```
> avendesora help -b accounts
```

When things go wrong, you can use *log command* to quickly view the log file:

```
> avendesora log
```

The logfile is kept in the `~/config/avendesora` directory and this command opens it directly in your editor. It can be very helpful in debugging account discovery issues.

At this point you should have *initialized your accounts* and *configured your window manager* and done the *initial configuration* of *Avendesora*.

### 3.3.1 Shell Account

In this example an account is provisioned to hold your Unix login password. You will not be able to use *Avendesora* to autotype your passcode when you login into your account, but you will be able to use it to enter the passcode when running shell commands like *sudo*.

To start, run the command to add an account. By default, three account templates are available. They are, in order of complexity: *shell*, *website*, and *bank*. The *shell* template assumes that there is only a passcode and any account discovery would be through the window title rather than by examining a URL.

To provision the new account use:

```
> avendesora add shell
```

Your editor should open with something that looks like this:

```
class NAME(Account):  
    desc = 'DESCRIPTION'  
    aliases = 'ALIAS1 ALIAS2'  
    passcode = Passphrase()
```

```
# Avendesora: Alternatively use PasswordRecipe('12 2u 2d 2s')
# Avendesora: or '12 2u 2d 2c!@#$$%' to specify valid symbol characters.
    discovery = RecognizeTitle(
        '_TITLE1_', '_TITLE2_',
        script='{passcode}{return}'
    )

# Avendesora: Tailor the account entry to suit you needs.
# Avendesora: You can add or delete class attributes as you see fit.
# Avendesora: The 'n' key should take you to the next field name.
# Avendesora: Use 'cw' to specify a field name, or delete it if unneeded.
# Avendesora: Fields surrounded by << and >> will be hidden.
# Avendesora: All lines that begin with '# Avendesora:' are deleted.
```

In this example it is assumed that your editor is Vim. You would jump to the first field by typing ‘n’ (next) and then modify the field by typing ‘cw’ (change word). In this example the first ‘n’ takes you to `_NAME_` and you would use ‘cw’ to change it to `LinuxLogin`. You should choose your account name carefully. Once set, you should never change an account name because it will result in the generated secrets associated with the account changing. If there is a chance that you might have more than one linux account, you should add more to the account name to make it unique. You can always provide a short easy to type alternative as an alias. For example, in this case the account username is `x57107048`, so you might want to add that to the account name to make it unique. Once you have entered the account name, hit ‘Esc’ to exit insert mode and type ‘n’ to go to the next field, `_DESCRIPTION_`. The account name is probably all the description we need, so you can simply delete this whole field by typing ‘dd’ (delete line). Moving on, you can replace the aliases with ‘login’ and ‘linux’. You can add additional aliases or delete the ones you don’t need. We will assume that you want to add your username, which was not anticipated by the template. To do so type ‘o’ to open a new line and type:

```
username = 'x57107048'
```

In general using passphrases is preferred to using passwords, the reason being that they are much easier to remember and type. That is important in this case because you will need to remember and enter your passcode when you login to your account, *Avendesora* cannot help you in that case. The template was configured to use a passphrase for the passcode, so no change is needed here.

Finally replace the titles with ‘sudo \*’. Once you have something that looks like this, you can exit the editor with ‘ZZ’:

```
class LinuxLogin(Account):
    aliases = 'linux login'
    username = 'x57107048'
    passcode = Passphrase()
# Avendesora: Alternatively use PasswordRecipe('12 2u 2d 2s')
# Avendesora: or '12 2u 2d 2c!@#$$%' to specify valid symbol characters.
    discovery = RecognizeTitle(
        'sudo *',
        script='{passcode}{return}'
    )

# Avendesora: Tailor the account entry to suit you needs.
# Avendesora: You can add or delete class attributes as you see fit.
# Avendesora: The 'n' key should take you to the next field name.
# Avendesora: Use 'cw' to specify a field name, or delete it if unneeded.
# Avendesora: Fields surrounded by << and >> will be hidden.
# Avendesora: All lines that begin with '# Avendesora:' are deleted.
```

There is no need to delete the embedded *Avendesora* instructions, they are deleted automatically when you save the file.

If you were to immediately edit the account again with:

```
> avendesora edit linuxlogin
```

you should see something like this:

```
class LinuxLogin(Account):
    aliases = 'linux login'
    username = 'x57107048'
    passcode = Passphrase()
    discovery = RecognizeTitle(
        'sudo *',
        script='{passcode}{return}'
    )
```

Notice that all the *Avendesora* instructions were removed.

You can show all the values associated with this account using the *values* command:

```
> avendesora values LinuxLogin
names: linuxlogin, linux, login
passcode: <reveal with 'avendesora value linuxlogin passcode'>
username: x57107048
```

Notice that the passcode is considered secret, so *Avendesora* does not actually show it when displaying all of the values. To see it, use:

```
> avendesora value LinuxLogin passcode
passcode: wigwam mistrust afflict refit
```

The value command will also write the secret directly to the clipboard:

```
> avendesora value --clipboard LinuxLogin passcode
```

By default *Avendesora* is configured to use the primary clipboard. You use the middle mouse button to paste from the primary clipboard. You can also modify the *xsel\_executable* setting to modify this behavior.

You can also write directly to the standard output (normally *Avendesora* writes to the TTY so that it can erase any secrets after a minute has elapsed). In this way you can use *Avendesora* within shell scripts (but you should consider rewriting your script in Python using the *Avendesora API*):

```
> pw value -s login 'user="{username}:{passcode}"' | curl -K - https://mywork.com/~
↪x57107048/latest
```

In this example, I needed to create an arbitrary string containing the username and password, so I combined *Avendesora's* *script* feature with the `-stdout` (`-s`) option to produce and pass the needed string to curl through a pipe.

You can also have *Avendesora* attempt to show you your *login credentials* for the account using:

```
> avendesora login LinuxLogin
username: x57107048
passcode: wigwam mistrust afflict refit
```

To show the login credentials *Avendesora* looks for candidate usernames (username, email) and candidate passcodes (passcode, password, passphrase).

## Short Cuts

*Avendesora* offers many ways to allow you to reduce or simplify your typing. In particular:

1. The account name is case insensitive:

```
> avendesora login linuxlogin
username: x57107048
passcode: wigwam mistrust afflict refit
```

2. You can give an alias rather than the account name:

```
> avendesora login linux
username: x57107048
passcode: wigwam mistrust afflict refit
```

3. Many of the command names have single letter abbreviations:

```
> avendesora l linux
username: x57107048
passcode: wigwam mistrust afflict refit
```

4. On the *value command*, if you do not specify a field, it will offer the passcode, password, or passphrase if available:

```
> avendesora v linux
passcode: wigwam mistrust afflict refit
```

5. If there is one argument and it is not recognized as a command name, it is treated as the account name and your login credentials are displayed:

```
> avendesora linux
username: x57107048
passcode: wigwam mistrust afflict refit
```

6. If there is more than one argument and the first is not recognized as a command name, it is treated as the account name and the *value command* is run:

```
> avendesora linux username
x57107048
```

7. Finally, people often alias 'pw' to 'avendesora' in their shell to make running *Avendesora* easier:

```
> pw linux
username: x57107048
passcode: wigwam mistrust afflict refit
```

## Auto Entry

You *LinuxLogin* account was provisioned with account discovery by way of the window title. This assumes that your shell adds the currently running command to the window title. Most shells are configured to do this by default, or can be configured to do so, though it may take some digging on the web to find the magic incantation to do so. Notice that one window title was given: 'sudo \*'. This matches a sudo command with arguments ('\*' is a wildcard character that matches any string of characters). To try out the account discovery, type:

```
> sudo make me a sandwich
[sudo] password for x57107048: <Alt-p>
```

Here <Alt-p> indicates that you should type your *Avendesora* hot key (hopefully you *set this up earlier*). It should run ‘avendesora value’. Since no account was given with this command, *Avendesora* attempts to discover which account should be used. It does so by offering the window title to each account provisioned with account discovery to see which account it matches. Assume it only matches LinuxLogin. Then the corresponding discovery script is run, in which case is ‘{passcode}{return}’. This script simulates the keyboard and types the passcode and then types the enter key, which should authenticate you with sudo and allow the command to run. If the window title matches several accounts, then each is offered up in a selection box and you choose the one you want (use ‘j’ and ‘k’ to navigate to desired section and ‘Enter’ to select or ‘Esc’ to cancel).

### 3.3.2 Website Account

In this example an account is provisioned to hold information typical to a website:

```
> avendesora add website
```

Your editor should open with something that looks like this:

```
class NAME (Account):
    desc = 'DESCRIPTION'
    aliases = 'ALIAS1 ALIAS2'
    username = 'USERNAME'
    email = 'EMAIL'
    passcode = PasswordRecipe('12 2u 2d 2s')
# Avendesora: length is 12, includes 2 upper, 2 digits and 2 symbols
# Avendesora: Alternatively use '12 2u 2d 2c!@#$$&' to specify valid symbol characters.
# Avendesora: Alternatively use Passphrase()
    questions = [
        Question("QUESTION1?"),
        Question("QUESTION2?"),
        Question("QUESTION3?"),
    ]
    urls = 'URL'
# Avendesora: specify urls if there are multiple recognizers.
    discovery = RecognizeURL(
        'URL',
        script='{email}{tab}{passcode}{return}'
    )
# Avendesora: Specify list of urls to recognizer if multiple pages need same script.
# Avendesora: Specify list of recognizers if multiple pages need different scripts.

# Avendesora: Tailor the account entry to suit you needs.
# Avendesora: You can add or delete class attributes as you see fit.
# Avendesora: The 'n' key should take you to the next field name.
# Avendesora: Use 'cw' to specify a field name, or delete it if unneeded.
# Avendesora: Fields surrounded by << and >> will be hidden.
# Avendesora: All lines that begin with '# Avendesora:' are deleted.
```

Use ‘n’ to step through the various fields and ‘cw’ to change the field. You can delete any fields that you do not need, or add any that you do. Here is an example of what it might look like when filled out completely after the instructions have been removed:

```
class Elevate84932153377 (Account):
    desc = 'Virgin America frequent flier plan'
```

```

aliases = 'elevate virgin virginamerica'
phone = '1.877.FLY.VIRGIN'
account = '8493-215-3377'
email = 'perrin.aybara@gmail.com'
passcode = PasswordRecipe('12 2u 2d 2s')
questions = [
    Question('mothers maiden name?'),
    Question('fathers middle name?'),
]
urls = 'https://www.virginamerica.com/cms/elevate-frequent-flyer'
discovery = RecognizeURL(
    'https://virginamerica.com',
    'https://www.virginamerica.com',
    script='{email}{tab}{passcode}{return}'
)

```

Notice that a very specific name was given to the account. This was done to allow additional Elevate accounts to be created, which might be needed for other family members or in case your account was ever compromised. Once you generate secrets from an account it is important that you not change the account name as that will change the values used for the secrets. Thus, if you choose a very selective account name you are less likely to need to change its name in the future. Of course, that name would be difficult to type, so you should give simpler names in the account aliases.

You can specify any information you feel is appropriate. Generally that includes the account number and the email you gave when creating the account.

You can give your passcode as password using `PasswordRecipe`. In this case you give a string that describes the characteristics of the password you want. The first value is the length of the password (12 characters), and then number of required characters of each type (2 upper case, 2 digits, and 2 symbols). If you are restricted to a specific set of symbols, such as `+=_-`, you can use `'2c+=_-'` to signify that two of the specified characters should be included (ex: `PasswordRecipe('12 2u 2d 2c+=_-')`). Alternatively, you can specify `Passphrase()` like in the shell account above. Or, you can explicitly specify the password. In this case you should indicate that the value is a secret so it is somewhat protected. There are two ways of doing that.

1. You specify the password as an argument to `Hide()`. Example: `Hide('catch22')`. In this case *Avendesora* protects the value as a secret, but it will show up unconcealed when viewing your account file.
2. You can specify the password embedded in `<<` and `>>`. For example: `<<catch22>>`. If you do that, the value is converted to base64 and passed as an argument to `Hidden()`. Thus, when you view the account file you will see: `Hidden("Y2F0Y2gyMg==")`. This makes it harder for anybody that happens to glance over your shoulder while you have your account file open to recognize and remember your password. In this case the encoded password is not encrypted, and it is easy to recover using *Avendesora's* `reveal` command or the linux `base64` command.

Many websites ask 'security' questions. These questions represent a back door into your account. If you forget your password, you can access your account by answering these questions. However, anybody else that happens to know the answers to these questions, such as your evil twin, can also use them to access your account. *Avendesora* defeats your evil twin by generating completely random answers to these personal questions. By default, `Question()` takes a string and turns it into three random words (be careful not to change the string after you have given the website the answers; doing so changes the answers). You can specify as many questions needed.

If you are not free to give arbitrary answers to your questions, such as if the website gives you a small set of acceptable answers, then you can give the answer along with the question:

```

questions = [
    Question('favorite subject in school?', answer=<<recess>>)),
    Question('favorite composer?' answer=<<chuck berry>>)),
]

```

Lastly this account sets up the web interface by specifying `urls` and `discovery`. The `urls` field is used by the `browse`

*command*, which opens your browser and navigates to the login page. For example:

```
> avendesora browse virgin
```

This can generally be done directly from your window manager, allowing you to open your account without needing to use a shell. In Gnome you can do so with Alt-F2 (Run Command). You can get the same functionality from other window managers by installing and assigning *dmenu* to a keyboard shortcut.

If you use the *browse command* on an account without a *urls* field, *Avendesora* will try to find one in the *discovery* field (it looks for URLs given to an instance of *RecognizeURL*, however it can get complicated if there is more than one instance of *RecognizeURL*. In such cases it is generally better to explicitly specify *urls*).

The *discovery* field is used to recognize that this is the account to use when *Avendesora* is asked to login into the *virginamerica.com* site. Notice that several URLs are given to *RecognizeURL()*, this is necessary when the website allows you to login using different domain names. *RecognizeURL()* is a variant of *RecognizeTitle()* that is attuned to the titles generated by browsers that have been configured to place the URL in the window title bar. This makes it more robust in this particular case. Also notice that the expected protocol (*https*) is given with the URLs. In this way, *Avendesora* will refuse to send your login credentials if the connection is not encrypted using the *https* protocol. The final argument to *RecognizeURL()* is the script that logs you in. In this case the script specifies that the value of the email field should be entered into the browser, followed by a tab, then the passcode, then a return.

It is possible to configure account discovery to support several secrets. To do so, place the recognizers in a list and specify different scripts for each. For example, many websites ask you to answer your security questions in order to confirm you are really you. This becomes easier with:

```
discovery = [  
  RecognizeURL(  
    'https://virginamerica.com',  
    'https://www.virginamerica.com',  
    script='{email}{tab}{passcode}{return}',  
    name='login'  
  ),  
  RecognizeURL(  
    'https://virginamerica.com',  
    'https://www.virginamerica.com',  
    script='{questions}{return}'  
    name='challenge question'  
  ),  
]
```

In this case if you trigger *Avendesora* (using *Alt-p*) while on the Virgin America website, it will respond by asking you if you want to login or answer a challenge question (in this case both recognizers trigger, forcing the choice). You can give different URLs for each case so that the choice is made automatically for you:

```
discovery = [  
  RecognizeURL(  
    'https://www.virginamerica.com/cms/elevate-frequent-flyer',  
    script='{email}{tab}{passcode}{return}',  
    name='login'  
  ),  
  RecognizeURL(  
    'https://www.virginamerica.com/cms/challenge',  
    script='{questions}{return}'  
    name='challenge question'  
  ),  
]
```



### 3.3.3 Bank Account

Bank accounts are similar to web accounts, but generally contain multiple account numbers and even more secrets. Create a bank account using:

```
> avendesora add bank
```

After you edit the various fields you may end up with something like this:

```
class MechanicsBank(Account):
    aliases = 'mb bank'
    username = Passphrase(length=2, sep='_')
    email = 'brandelwyn.alVere@aol.com'
    checking = <<008860636145>>,
    savings = <<029370021509>>,
    creditcard = <<5251014820644156>>,
    ccv = <<588>>
    expiration = <<03/2020>>
    ccn = Script('{account.creditcard}{tab}{ccv}{tab}')
    passcode = PasswordRecipe('16 2u 2l 2d 2c#%=_-<>')
    verbal = Passphrase(length=2)
    questions = [
        Question('mothers maiden name?'),
        Question('fathers middle name?'),
    ]
    routing = '013521325'
    customer_support = ''
        credit cards: 800-730-6259
        banking: 800-861-5715
    ''
    urls = 'https://secure.mechanicsbank.com/login'
    discovery = RecognizeURL(
        'https://mechanicsbank.com',
        'https://www.mechanicsbank.com',
        'https://secure.mechanicsbank.com',
        'https://online.mechanicsbank.com',
        script='{username}{tab}{passcode}{return}'
    )
```

In this case, since this account holds real money, a bit more attention is given to security. For example, the username was specified as a 2 word passphrase, making very unlikely that anyone could guess your username. Furthermore, your account numbers and your credit-cards CCV number are hidden by decorating them with << >> (you could also just use Hide()).

Also, a verbal password is included. Many financial institutions allow you to set up a verbal password that you use when calling in. This is an important protection in that it stops people that know you well, such as your evil twin, from calling in and impersonating you. A short passphrase is perfect for this use as it is easy to communicate to someone over the phone.

In this example separate fields are used for each account number. If you have access to the accounts of several people, for example you and your children, you might use a dictionary for the accounts of each person, as follows:

```
brandelwyn = dict(
    checking = <<008860636145>>,
    savings = <<029370021509>>,
    creditcard = <<5251-0148-2064-4156>>,
)
marin = dict(
```

```
    checking = <<275137908190>>,
    savings = <<874647693848>>,
)
egwene = dict(
    checking = <<718467200674>>,
    savings = <<623691894130>>,
)
```

Now to get Egwene's checking account number you would use:

```
avendesora bank egwene.checking
```

Security questions and account discovery are handled as given above.

The *ccn* or credit card number field is given as a script. With this you can navigate to any website that needs your credit card number and CCV and enter it by typing:

```
<Alt-F2> avendesora bank ccn
```

Here `<Alt-F2>` is assumed to be the hot key sequence that runs a shell command directly from the window manager (Gnome uses `Alt-F2`, but yours may be different). Doing so causes your credit card number, followed by a tab, followed by your CCV, and followed by another tab to be typed into the page. You could conceivably start by typing your name and follow with your address, but there is enough variability in websites that this would likely not work on all of them, so it is generally best to limit the script to a small number of the most helpful fields.

### 3.3.4 Finding Accounts

*Avendesora* provides two ways of finding account names if you do not remember them. First is the *find command*, which given a bit of text lists all of the accounts that contain that text in their names or their aliases. For example:

```
> avendesora find bank
bank-america (ba, boa, bofa) -- home mortgage
citibank-mastercard (mc, mastercard, citibank) -- credit card
mechanicsbank (mb bank) -- bank
```

The next is the *search command*, which given a bit of text lists all of the accounts that contain that text in any of the non-secret account values. For example:

```
> avendesora search bank
bank-america (ba, boa, bofa) -- home mortgage
capitalone (co, ing) -- savings bank
citibank-mastercard (mc, mastercard, citibank) -- credit card
mechanicsbank (mb bank) -- bank
wellsfargo (wf) -- old bank
```

In both cases the name of the account is listed first followed by the account aliases (within parentheses). The description, if available, is appended to the end.

### 3.3.5 Modifying Accounts

Once an account exists, it can be modified using the *edit command*:

```
> avendesora edit bank
```

This opens the MechanicsBank account in your editor (you can select your editor by modifying the *edit\_account setting*). Once you modify your account, you should save the file and exit the editor. The change will be checked and if there are any errors, you will be given a chance to reopen the account file and fix the problem.

### 3.3.6 Additional Features

In addition what has already been introduced, *Avendesora* provides a collection of advanced features. Those include ...

- *Avendesora* supports a wide variety of types of secrets, including support for *one-time passwords*. These secrets are described in *Account Helpers*.
- The *archive* and *changed* commands provide an ability to create a backup copy of all your passwords. These command are described in the section on *upgrading*.
- Two techniques that provide an extra measure of security for accounts are *stealth accounts* and *misdirection*.
- *Avendesora* provides several ways that help protect you from *phishing*. You should be aware of these methods and make sure you use them.
- *Avendesora* allows you to share master seeds with a partner, and once done allow you to easily and securely create new shared secrets. This is described in the section on *collaboration*.
- Once you share a master seed, you can use the *identity command* as described in *confirming identity* to securely verify that you are communicating with your partner.
- You can quickly print out the *NATO phonetic alphabet*, which can be useful when trying to communicate complex character sequences over the phone.

## 3.4 Accounts

Account information is stored in account files. The list of account files is given in `~/.config/avendesora/accounts_files`. New account files are created using `'avendesora new'`, but to delete an accounts file, you must manually remove it from `accounts_files`. Once an accounts file exists, you may add accounts to it using `'account add'`. Use the `-f` option to specify which file is to contain the new account. Modifying or deleting an account is done with `'account edit account_name'`. To delete the account, simply remove all lines associated with the account.

An account is basically a collection of attributes organized as a subclass of the Python `avendesora.Account` class. For example:

```
class ManetherenTimes(Account):
    aliases = 'times mt'
    username = 'nynaeve'
    email = 'nynaeve@gmail.com'
    passcode = PasswordRecipe('12 2u 2d 2s')
    discovery = RecognizeURL(
        'https://myaccount.manetherentimes.com',
        script='{email}{tab}{passcode}{return}'
    )
```

One creates an account using:

```
> avendesora add <type>
```

where `<type>` is either *shell*, *website* or *bank*. Choose the template that seems most appropriate (see *overview* and *add command* for more information) and edit it to your needs.

If after configuring your account you feel the need to change it, you can use the *edit command* to do so:

```
> avendesora edit manetherentimes
```

The account name is case insensitive and can be replaced by one of the aliases. Once created, most of the field values can be retrieved simply by asking for them. For example:

```
> avendesora value times username
username: nynaeve
```

In general, values can be strings, arrays, dictionaries, and special Avendesora classes. For example, you could have an array of security questions:

```
questions = [
    Question("What is your mother's maiden name?"),
    Question("What city were you born?"),
    Question("What is first pet's name?"),
]
```

Then you can request the answer to a particular question using its index:

```
> avendesora value times questions.0
questions.0 (What is your mother's maiden name?): portrayal tentacle fanlight
```

*questions* is the default array field, so you could have shortened your request by using '0' rather than 'questions.0'. You might be thinking, hey, that is not my mother's maiden name. That is because *Question* is a 'generated secret'. It produces a completely random answer that is impossible to predict. Thus, even family members cannot know the answers to your security questions.

A dictionary is often used to hold account numbers:

```
class TwoRiversCU(Account):
    accounts = {
        'checking': '1234-56-7890',
        'savings': '0123-45-6789',
    }
```

You then access its values using:

```
> avendesora value tworiverscu accounts.checking
accounts.checking: 1234-56-7890
```

You might consider your account numbers as sensitive information. In this case you can hide them with:

```
class TwoRiversCU(Account):
    accounts = {
        'checking': Hide('1234-56-7890'),
        'savings': Hide('0123-45-6789'),
    }
```

Doing so means that *Avendesora* will try to protect them from accidental disclosure. For example, it will attempt to erase the screen after displaying them for a minute. You may also be concerned with someone looking over your shoulders when you are editing your accounts file and stealing your secrets. To reduce the chance, you can encode the secrets:

```
class TwoRiversCU(Account):
    accounts = {
        'checking': Hidden('MTIzNC01Ni03ODkw'),
    }
```

```
'savings': Hidden('MDEyMy00NS02Nzg5'),
}
```

The values are now hidden, but not encrypted. They are simply encoded with base64. Any knowledgeable person with the encoded value can decode it back to its original value. Using `Hidden` makes it harder to recognize and remember the value given only a quick over-the-shoulder glance. It also marks the value as sensitive, so it will only be displayed for a minute. You generate the encoded value using the *conceal command*.

If this is not enough security, you can encrypt the values and access them using `avendesora.GPG` or `avendesora.Scrypt`.

You can find the specifics of how to specify or generate your secrets in *Account Helpers*.

Any value that is an instance of the `avendesora.GeneratedSecret` class (`avendesora.Password`, `avendesora.Passphrase`, ...) or the `avendesora.ObscuredSecret` class (`avendesora.Hide`, `avendesora.Hidden`, `avendesora.GPG`, ...) is considered sensitive. It is only given out in a controlled manner. For example, running the *values command* displays all fields, but the values that are sensitive are replaced by instructions on how to view them. They can only be viewed individually:

```
> avendesora values times
names: manetherentimes, times, mt
email: nynaeve@gmail.com
passcode: <reveal with 'avendesora value manetherentimes passcode'>
username: nynaeve
```

Notice the passcode is not shown. You can circumvent this protection by adding `is_secret=False` to the argument list of the secret.

The *aliases* and *discovery* fields are not shown because they are considered tool fields (see *Account Discovery* for more information on discovery). Other tool fields include *NAME*, *default*, *master*, *browser*, and *default\_url*. *default* is the name of the default field, which is the field you get if you do not request a particular field. Its value defaults to *password*, *pasphrase*, or *passcode* (as set by *default\_field* setting), but it can be set to any account attribute name or it can be a *script*. *browser* is the default browser to use when opening the account, run the *browse command* to see a list of available browsers.

The value of *passcode* is considered sensitive because it is an instance of `PasswordRecipe`, which is a subclass of `GeneratedSecret`. If you wish to see the *passcode*, use:

```
> avendesora value mt
passcode: TZuk8:u7qY8%
```

This value will be displayed for a minute and is then hidden. If you would like to hide it early, simply type Ctrl-C.

An attribute value can incorporate other attribute values through use of the `avendesora.Script` class as described in *Scripts*. For example, consider an account for your wireless router that contains the following:

```
class EmondsFieldInnWifi(Account):
    aliases = 'wifi'
    ssid = {
        'emonds_field_inn_guests': Passphrase(),
        'emonds_field_inn_private': Passphrase(),
    }
    guest = Script('SSID: emonds_field_inn_guests{return}password: {ssid.emonds_field_
↪inn_guests}')
    private = Script('SSID: emonds_field_inn_private{return}password: {ssid.emonds_
↪field_inn_private}')
```

The *ssid* field is a dictionary that contains the SSID and passphrases for each of the wireless networks provided by the router. This is a natural and compact representation for this information, but accessing it as a user in this form requires

two steps to access the information, one to get the SSID and another to get the passphrase. This issue is addressed by adding the `guest` and `private` attributes. The `guest` and `private` attributes are scripts that gives the SSID and interpolate the passphrase. Now both can easily accessed at once with:

```
> avendesora value wifi guest
SSID: emonds_field_inn_guests
password: delimit ballcock fibber levitate
```

Use of *Avendesora* secrets classes (`avendesora.GeneratedSecret` or `avendesora.ObscuredSecret`) is confined to the top two levels of account attributes, meaning that they can be the value of the top-level attributes, or the top-level attributes may be arrays or dictionaries that contain objects of these classes, but it can go no further.

It is important to remember that generated secrets use the account name and the field name when generating their value, so if you change the account name or field name you will change the value of the secret. For this reason is it important to choose a good account and field names up front and not change them. It should be very specific to avoid conflicts with similar accounts created later. For example, rather than choosing *Gmail* as your account name, you might want to include your username, ex. *GmailThomMerrilin*. This would allow you to create additional gmail accounts later without ambiguity. Then just add *gmail* as an alias to the account you use most often.

Account and field names are case insensitive. So you can use *Gmail* or *gmail*. Also, if the account or field names contains an underscore, you can substitute a dash. So if the account name is *Gmail\_Thom\_Merrilin*, you can use *gmail-thom-merrilin* instead.

## 3.5 Advanced Usage

### 3.5.1 Avoiding Phishing Attacks

Phishing is a very common method used on the web to get people to unknowingly divulge sensitive information such as account credentials. It is generally accomplished by sending misleading URLs in email or placing them on websites. When you visit these URLs you are taken to a site that looks identical to the site you were expecting to go to in the hope that you are tricked into giving up your account credentials. It used to be that if you carefully inspected the URL you could spot deception, but even that is no longer true.

*Avendesora* helps you avoid phishing attacks in two ways. First, you should never go to one of your secure sites by clicking on a link. Instead, you should use *Avendesora*'s *browse command*:

```
avendesora browse chase
```

In this way you use the URL stored in *Avendesora* rather than trusting a URL link provided by a third party. Second, you should auto-enter the account credentials using *Avendesora*'s account discovery based on `avendesora.RecognizeURL` (be sure to use `avendesora.RecognizeURL` for websites rather than `avendesora.RecognizeTitle` when configuring account discovery, `avendesora.RecognizeURL` is not fooled by phishing sites).

### 3.5.2 Account Discovery

If you do not give an account to '*avendesora value*', *Avendesora* tries to determine the account by simply asking each account if it is suitable. An account can look at the window title, the user name, the host name, the working directory, and the environment variables to determine if it is suitable. If so, it nominates itself. If there is only one account nominated, that account is used. If there are multiple nominees, then a small window pops up allowing you to choose which account you wish to use.

To configure an account to trigger when a particular window title is seen, use:

```
discovery = RecognizeTitle(
    'Chase Online *',
    script='{username}{tab}{passcode}{return}'
)
```

The title is a glob string, meaning that ‘\*’ matches any combination of characters. The script describes what *Avendesora* should output when there is a match. In this case it outputs the username field, then a tab, then the passcode field, then a return (see *Scripts*).

Matching window titles can be fragile, especially for websites because the titles can vary quite a bit across the site and over time. To accommodate this variation, you can give multiple glob patterns:

```
discovery = RecognizeTitle(
    'CHASE Bank*',
    'Chase Online*',
    script='{username}{tab}{passcode}{return}'
)
```

However, in general, it is better to match the URL. This can be done in Firefox and Chrome by adding extensions that place the URL in the window title and then using *avendesora.RecognizeURL* to do the recognition.

If you use Firefox, you should install the [Add URL to Window Title](#) extension. It is a plugin that makes discovery easier and more robust by adding the URL to the title. For *Chrome* the appropriate plugin is [URL in Title](#). It is recommended that you install the appropriate one into your browser. For *Add URL To Window Title*, set the following options:

```
show full URL = yes
separator string = '-'
show field attributes = no
```

For *URL in Title*, set:

```
tab title format = '{title} - {protocol}://{hostname}{port}/{path}'
```

If you use *qutebrowser* as your browser, you should add the following to your `~/.config/qutebrowser/config.py` file:

```
c.window.title_format = '{title} - {current_url} - qutebrowser'
```

*avendesora.RecognizeURL* is designed to recognize such titles. Once you have deployed the appropriate plugin, you can use:

```
discovery = RecognizeURL(
    'https://chaseonline.chase.com',
    'https://www.chase.com',
    script='{username}{tab}{passcode}{return}'
)
```

When giving the URL, anything specified must match and globbing is not supported. If you give a partial path, by default *Avendesora* matches up to what you have given, but you can require an exact match of the entire path by specifying `exact_path=True` to *avendesora.RecognizeURL*. If you do not give the protocol, the `default_protocol` (https) is assumed.

In general you should use *avendesora.RecognizeURL* rather than *avendesora.RecognizeTitle* for websites if you can. Doing so helps protect you from phishing attacks by carefully examining the URL.

When account discovery fails it can be difficult to determine what is going wrong. When this occurs, you should first examine the log file:

```
> avendesora log
```

It should show you the window title and the recognized title components. You should first assure the title is as expected. If *Add URL to Window Title* or *URL in Title* generated the title, then the various title components should also be shown. Then run *Avendesora* as follows:

```
> avendesora value --verbose --title '<title>'
```

The title should be copied from the log file. The verbose option causes the result of each test to be included in the log file, so you can determine which recognizer is failing to trigger. You can either specify the verbose option on the command line or in the config file.

## Recognizers

The following recognizers are available:

```
RecognizeAll(<recognizer>..., [script=<script>])
RecognizeAny(<recognizer>..., [script=<script>])
RecognizeTitle(<title>..., [script=<script>])
RecognizeURL(<title>..., [script=<script>, [name=<name>,]] [exact_path=<bool>])
RecognizeHost(<host>..., [script=<script>])
RecognizeUser(<user>..., [script=<script>])
RecognizeCWD(<cwd>..., [script=<script>])
RecognizeEnvVar(<name>, <value>, [script=<script>])
RecognizeNetwork(<mac>..., [script=<script>])
RecognizeFile(<path>, [<contents>,] [<ttl>,] [script=<script>])
```

*avendesora.RecognizeAll* and *avendesora.RecognizeAny* can be used to combine several recognizers. For example:

```
discovery = RecognizeAll(
    RecognizeTitle('sudo *'),
    RecognizeUser('hhyde'),
    script='{passcode}{return}'
)
```

If the recognizers are given in an array, all are tried, and each that match are offered. For example:

```
discovery = [
    RecognizeURL(
        'http://www.querty-forum.org',
        script='admin{tab}{passcode}{return}',
        name='admin',
    ),
    RecognizeURL(
        'http://www.querty-forum.org',
        script='thecaretaker{tab}{passcode}{return}',
        name='thecaretaker',
    ),
]
```

In this case, both recognizers recognize the same URL, thus they are both be offered for this site. But each has a different script. The name allows the user to distinguish the available choices.

If there is a need to distinguish URLs where is one is a substring of another, you can use *exact\_path*:



```
discovery = [
    RecognizeURL(
        'https://mybank.com/Authentication',
        script='{username}{return}',
        exact_path=True,
    ),
    RecognizeURL(
        'https://mybank.com/Authentication/Password',
        script='{passcode}{return}',
        exact_path=True,
    ),
]
```

`avendesora.RecognizeFile` checks to determine whether a particular file has been created recently. This can be used in scripts to force secret recognition. For example, the titles used by Firefox and Thunderbird when collecting the master password is either non-existent or undistinguished. These programs also produce a large amount of uninteresting chatter on their output, so it is common to write a shell script to run the program that redirects their output to `/dev/null`. Such a script can be modified to essentially notify *Avendesora* that a particular password is desired. For example, for Thunderbird:

```
#!/bin/sh
touch /tmp/thunderbird-1024
/usr/bin/thunderbird > /dev/null
```

Here I have added my user id (`uid=1024`) to make the filename unique so I am less likely to clash with other users. Alternately, you could choose a path that fell within your home directory. Then, adding:

```
class Firefox(Account):
    desc = 'Master password for Firefox and Thunderbird'
    passcode = Password()
    discovery = RecognizeFile(
        '/tmp/thunderbird-1024', wait=60, script='{passcode}{return}'
    )
```

If the specified file exists and has been updated within the last 60 seconds, then secret is recognized. You can specify the amount of time you can wait in between running the script and running *Avendesora* with the `wait` argument, which takes a number of seconds. It defaults to 60.

Using this particular approach, every secret needs its own file. But you can share a file by specifying the file contents. Then the script could be rewritten as:

```
#!/bin/sh
echo thunderbird > ~/.avendesora-password-request
/usr/bin/thunderbird > /dev/null
```

Then you would add something like the following to your accounts file:

```
class Firefox(Account):
    desc = 'Master password for Firefox and Thunderbird'
    passcode = Password()
    discovery = RecognizeFile(
        '~/.avendesora-password-request',
        contents='thunderbird',
        script='{passcode}{return}'
    )
```

### 3.5.3 Security Questions

Security questions are form of security theater imposed upon you by many websites. The claim is that these questions increase the security of your account. In fact they often do the opposite by creating additional avenues of access to your account. Their real purpose is to allow you to regain access to your account in case you lose your password. If you are careful, this is not needed (you do back up your *Avendesora* accounts, right?). In this case it is better to randomly generate your answers.

Security questions are handled by adding something like the following to your account:

```
questions = [  
    Question('oldest aunt?'),  
    Question('title of first job?'),  
    Question('oldest uncle?'),  
    Question('savings goal?'),  
    Question('childhood vacation spot?'),  
]
```

The string identifying the question does not need to contain the question verbatim, an abbreviated version is sufficient as long as it allows you to distinguish the question. However, once set, you should not change the question in the slightest; doing so changes the generated answer.

The questions are given as an array, and so are accessed with an index that starts at 0. Thus, to get the answer to who is your 'oldest aunt', you would use:

```
> avendesora value <accountname> 0  
questions.0 (oldest aunt): ampere reimburse duster
```

You can get a list of your questions so you can identify which index to use with:

```
> avendesora values <accountname>  
...  
questions:  
  0: oldest aunt? <reveal with 'avendesora value <accountname> questions.0'>  
  1: title of first job? <reveal with 'avendesora value <accountname> questions.1'>  
  2: oldest uncle? <reveal with 'avendesora value <accountname> questions.2'>  
  3: savings goal? <reveal with 'avendesora value <accountname> questions.3'>  
  4: childhood vacation spot? <reveal with 'avendesora value <accountname>  
->questions.4'>  
...
```

Or you can use:

```
> avendesora values <accountname> questions  
avendesora error: questions:  
  composite value found: questions:  
    0: oldest aunt?  
    1: title of first job?  
    2: oldest uncle?  
    3: savings goal?  
    4: childhood vacation spot?
```

By default, *Avendesora* generates a response that consists of 3 random words. This makes it easy to read to a person over the phone if asked to confirm your identity. Occasionally you will not be able to enter your own answer, but must choose one that is offered to you. In this case, you can specify the answer as part of the question:

```
questions = [  
    Question('favorite fruit?', answer='grapes'),
```

```

    Question('first major city visited?', answer='paris'),
    Question('favorite subject?', answer='history'),
]

```

When giving the answers you may want to conceal them to protect them from casual observation.

### 3.5.4 Opening Accounts in your Browser

*Avendesora* provides the *browse* command to allow you to easily open the website for your account in your browser. To do so, it needs two things: a URL and a browser.

#### Selecting the URL

*Avendesora* looks for URLs in the *urls* and *discovery* account attributes, with *urls* being preferred if both exist. *urls* may either be a string, a list, or a dictionary. If it is a string, it is split at white spaces to make it a list. If *urls* is a list, the URLs are considered unnamed and the first one given is used. If it a dictionary, the URLs are named. When named, you may specify the URL you wish to use by specifying the name to the *browse* command. For example, consider a *urls* attribute that looks like this:

```

class Dragon(Account):
    username = 'rand'
    passcode = Passphrase()
    urls = dict(
        email = 'https://webmail.dragon.com',
        vpn = 'https://vpn.dragon.com',
    )
    default_url = 'email'

```

You would access *vpn* with:

```
avendesora browse dragon vpn
```

By specifying *default\_url* you indicate which URL is desired when you do not explicitly specify which you want on the *browse* command. In this way, you can access your email with either of the following:

```

avendesora browse dragon email
avendesora browse dragon

```

If *urls* is not given, *Avendesora* looks for URLs in *avendesora.RecognizeURL* members in the *discovery* attribute. If the *name* argument is provided to *avendesora.RecognizeURL*, it is treated as a named URL, otherwise it is treated as an unnamed URL.

If named URLs are found in both *urls* and *discovery* they are all available to *browse* command, with those given in *urls* being preferred when the same name is found in both attributes.

#### Selecting the Browser

You can configure browsers for use by *Avendesora* using the *browsers* setting. By default, *browsers* contains the following:

```

browsers = dict(
    f = 'firefox -new-tab {url}',
    fp = 'firefox -private-window {url}',
    c = 'google-chrome {url}',

```

```
ci = 'google-chrome --incognito {url}',
q = 'qutebrowser {url}',
t = 'torbrowser {url}',
x = 'xdg-open {url}',
)
```

Each entry pairs a key with a command. The command will be run with *{url}* replaced by the selected URL when the browser is selected. You can choose which browser is used by specifying the *-browser* command line option on the *browse* command, by adding the *browser* attribute to the account, or by specifying the *default\_browser* setting in the *config file*. If more than one is specified, the command line option dominates over the account attribute, which dominates over the setting. By default, the default browser is *x*, which uses the default browser for your account.

### 3.5.5 One-Time Passwords

One-time passwords are often used as a second factor to provide an additional level of protection. They are especially useful when you are concerned about keyloggers.

*Avendesora* supports time-based one-time passwords (TOTP) that are fully compatible with, and can act as an alternative to or a replacement for, the *Google Authenticator* or *Authy* apps.

When first enabling one-time passwords with *Google Authenticator* you are generally presented with a QR code. Also included is a string of characters that are often referred to as the backup code. You would provide this string of characters to the OTP class to configure an account for a one-time password. For example, here is an account that requests your username and password on one page, and your one time password on another:

```
class AndorSavings (Account) :
    email = 'lini.eltring@yahoo.com'
    passcode = PasswordRecipe('16 2u 2d 2s')
    otp = OTP('JBSWY3DPEHPK3PXP')
    credentials = 'email passcode otp'
    urls = 'https://www.andorsavings.com/login.html'
    discovery = [
        RecognizeURL(
            'https://www.andorsavings.com/login.html',
            script='{email}{tab}{passcode}{return}',
            name='email & password',
        ),
        RecognizeURL(
            'https://www.andorsavings.com/googleVerify.html',
            script='{otp}{return}',
            name='authentication token',
        ),
    ]
]
```

Or, if you are lucky enough that they allow you to enter the OTP on the same page as your username and password, you might have:

```
class AndorSavings (Account) :
    email = 'lini.eltring@yahoo.com'
    passcode = PasswordRecipe('16 2u 2d 2s')
    otp = OTP('JBSWY3DPEHPK3PXP')
    credentials = 'email passcode otp'
    discovery = RecognizeURL(
        'https://www.andorsavings.com/login.html',
        script='{email}{tab}{passcode}{tab}{otp}{return}',
        name='email, passcode and authentication token',
    )
]
```

In this case, you only need one recognizer and specifying *urls* is no longer necessary because you only have one URL in the account.

This account adds a one time password as *otp*. It adds a *credentials* field that adds the one-time password to the output of the *credentials command*. It also adds a URL recognizer to allow semiautomatic entry of the one-time password to the browser. And finally it adds the *urls* to specify which URL the *browse command* should use.

It is easy to mimic *Google Authenticator*. Mimicking *Authy* is more difficult. To do so, follow [these instructions](#). Basically, the idea is to install the *Authy Chrome* app, start it, open the desired account, then back in *Chrome* open `chrome://extensions`, select *Developer Mode*, then click on 'Inspect views: main.html', search for *totp* function, set a break point in that function and wait until it trips, then copy the value of the *e* argument (a 32 digit hexadecimal number) to *hex\_seed* in the code below:

```
#!/usr/bin/env python3

from base64 import b32encode, b32decode
from pyotp import TOTP
from time import sleep

def int_to_bytes(x):
    return x.to_bytes((x.bit_length() + 7) // 8, 'big')

hex_seed = 0xNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
seed = b32encode(int_to_bytes(hex_seed))
print('SEED: %s' % seed)

otp = TOTP(seed, interval=10, digits=7)
print(otp.now())
sleep(10)
print(otp.now())
sleep(10)
print(otp.now())
sleep(10)
print(otp.now())
sleep(10)
print(otp.now())
```

Substitute your number for *NNN...NN*. Then run the script to display the seed or shared secret. It will also show five codes, one every 10 seconds. Every other code should match the value produced by the *Chrome* app. Be aware that every *Authy* app has its own seed, so the sequence that *Chrome* generates will be different from the sequence generated by your phone app or even a different *Chrome* app, and that is true even if they are generating tokens for the same account.

Once you are convinced that your seed is correct, add something like the following to your account to generate the one-time password:

```
otp = OTP('UM0HJVLT4HVWJQJC47Q8YXX4TU=====', interval=10, digits=7)
```

The *interval* and *digits* are specific to *Authy*.

### 3.5.6 Scripts

Scripts are strings that contain embedded account attributes. For example:

```
'username: {username}, password: {passcode}'
```

When processed by *Avendesora* the attributes are replaced by their value from the chosen account. For example, this script might be rendered as:

```
username: rand_alThor, password: R7ibHyPjWtG2
```

You can specify a script directly to the *value command*. You can specify them as account attributes (in this case then need to be embedded in *avendesora.Script*), or you can specify them to *account discovery recognizers*.

Scripts are useful if you need to combine an account value with other text, if you need to combine more than one account value, or if you want quick access to something that would otherwise need an additional key.

For example, consider an account for your wireless router, which might hold several passwords, one for administrative access and one or more for the network passwords. Such an account might look like:

```
class WiFi(Account):
    username = 'admin'
    passcode = Passphrase()
    networks = ["Occam's Router", "Occam's Router (guest)"]
    network_passwords = [Passphrase(), Passphrase()]
    privileged = Script('''
        SSID: {networks.0}
        password: {network_passwords.0}
    ''')
    guest = Script('''
        SSID: {networks.1}
        password: {network_passwords.1}
    ''')
    credentials = 'privileged guest username passcode'
```

Notice that *privileged* and *guest* were specified as scripts. Now the credentials for the privileged network are accessed with:

```
> avendesora value wifi privileged
SSID: Occam's Router
password: overdraw cactus devotion saying
```

You can also give a script rather than a field on the command line when running the *value command*:

```
> avendesora value scc '{username}: {passcode}'
rand_alThor: R7ibHyPjWtG2
```

For example, a place where this is useful is when specifying a username and password to curl:

```
> curl --user `avendesora value -s apache '{username}:{passcode}'` ...
```

It is also possible to specify a script for the value of the *default* attribute. This attribute allows you to specify the default field (which attribute name and key to use if one is not given on the command line). It also accepts a script rather than a field, but in this case it should be a simple string and not an instance of the *avendesora.Script* class. If you passed it as a *avendesora.Script*, it would be expanded before being interpreted as a field name, and so would result in a ‘not found’ error.

```
class SCC(Account):
    aliases = 'scc'
    username = 'rand_alThor'
    password = PasswordRecipe('12 2u 2d 2s')
    default = 'username: {username}, password: {password}'
```

You can access the script by simply not providing a field:

```
> avendesora value scc
username: rand_alThor, password: *m7Aqj=XBAs7
```

Finally, you pass a script to the account discovery recognizers. They specify the action that should be taken when a particular recognizer triggers. These scripts would also be simple strings and not instances of the *avendesora.Script* class. For example, this recognizer could be used to recognize Gmail:

```
discovery = [
  RecognizeURL(
    'https://accounts.google.com/ServiceLogin',
    'https://accounts.google.com/signin/v2/identifier',
    script='{username}{return}{sleep 2}{passcode}{return}'
    name='username and passcode',
  ),
  RecognizeURL(
    'https://accounts.google.com/signin/v2/sl/pwd',
    script='{passcode}{return}',
    name='passcode',
  ),
  RecognizeURL(
    'https://accounts.google.com/signin/challenge',
    script='{questions.0}{return}'
    name='challenge',
  ),
]
```

Besides the account attributes, you can use several other special attributes including: *{tab}*, *{return}*, and *{sleep N}*. *{tab}* is replaced by a tab character, *{return}* is replaced by a carriage return character, and *{sleep N}* causes a pause of N seconds. The sleep function is only active when auto-typing in account discovery.

### 3.5.7 Stealth Accounts

Normally *Avendesora* uses information from an account that is contained in an accounts file to generate the secrets for that account. In some cases, the presence of the account itself, even though it is contained within an encrypted file can be problematic. The mere presence of an encrypted file may result in you being compelled to open it. For the most damaging secrets, it is best if there is no evidence that the secret exists at all. This is the purpose of stealth accounts. (*Misdirection* is an alternative to stealth accounts).

The stealth accounts are predefined and have names that are descriptive of the form of the secret they generate, for example *word4* generates a 4-word pass phrase (also referred as the xkcd pattern):

```
> avendesora value word4
account: my_secret_account
gulch sleep scone halibut
```

The predefined accounts are kept in `~/config/avendesora/stealth_accounts`. You are free to add new accounts or modify the existing accounts.

Stealth accounts are subclasses of the *avendesora.StealthAccount* class. These accounts differ from normal accounts in that they do not contribute the account name to the secrets generators for use as a seed. Instead, the user is requested to provide the account name every time the secret is generated. The secret depends strongly on this account name, so it is essential you give precisely the same name each time. The term ‘account name’ is being use here, but you can enter any text you like. Best to make this text very difficult to guess if you are concerned about being compelled to disclose your GPG keys. You would not want your spouse simply try ‘ashleymadison’ after you walk away from your computer to gain access to your previously secret account.

The secret generator will combine the account name with the master seed before generating the secret. This allows you to use simple predictable account names and still get an unpredictable secret. The master seed used is taken from `master_seed` in the file that contains the stealth account if it exists, or the `user_key` if it does not. By default the stealth accounts file does not contain a master seed, which makes it difficult to share stealth accounts. You can create additional stealth account files that do contain master seeds that you can share with your associates.

### 3.5.8 Misdirection

One way to avoid being compelled to disclose a secret is to disavow any knowledge of the secret. However, the presence of an account in *Avendesora* that pertains to that secret undercuts this argument. This is the purpose of stealth accounts. They allow you to generate secrets for accounts for which *Avendesora* has no stored information. In this case *Avendesora* asks you for the minimal amount of information that it needs to generate the secret. However in some cases, the amount of information that must be retained is simply too much to keep in your head. In that case another approach, referred to as secret misdirection, can be used.

With secret misdirection, you do not disavow any knowledge of the secret, instead you say your knowledge is out of date. So you would say something like “I changed the password and then forgot it”, or “The account is closed”. To support this ruse, you must use the `-seed` (or `-S`) option to ‘avendesora value’ when generating your secret (secrets misdirection only works with generated passwords, not stored passwords). This causes *Avendesora* to ask you for an additional seed at the time you request the secret. If you do not use `-seed` or you do and give the wrong seed, you will get a different value for your secret. In effect, using `-seed` when generating the original value of the secret causes *Avendesora* to generate the wrong secret by default, allowing you to say “See, I told you it would not work”. But when you want it to work, you just interactively provide the correct seed.

You would typically only use misdirection for secrets you are worried about being compelled to disclose. So it behooves you to use an unpredictable additional seed for these secrets to reduce the chance someone could guess it.

Be aware that when you employ misdirection on a secret, the value of the secret stored in the archive will not be the true value, it will instead be the misdirected value.

### 3.5.9 Collaborating with a Partner

If you share an accounts file with a partner, then either partner can create new secrets and the other partner can reproduce their values once a small amount of relatively non-confidential information is shared. This works because the security of the generated secrets is based on the master seed, and that seed is contained in the accounts file that is shared in a secure manner once at the beginning. For example, imagine one partner creates an account at the US Postal Service website and then informs the partner that the name of the new account is *USPS* and the username is *taveren*. That is enough information for the second partner to generate the password and login. And notice that the necessary information can be shared over an insecure channel. For example, it could be sent in a text message or from a phone where trustworthy encryption is not available.

The first step in using *Avendesora* to collaborate with a partner is for one of the partners to generate and then share an accounts file that is dedicated to the shared accounts. This file contains the master seed, and it is critical to keep this value secure. Thus, it is recommended that the shared file be encrypted.

Consider an example where you, Siuan, are sharing accounts with your business partner, Moiraine. You have hired a contractor to run your email server, Elaida, who unbeknownst to you is reading your email in order to steal valuable secrets. Together, you and Moiraine jointly run Aes Sedai Enterprises. Since you expect more people will need access to the accounts in the future, you choose to name the file after the company rather than your partner. To share accounts with Moiraine, you start by getting Moiraine’s public GPG key. Then, create the new accounts file with something like:

```
avendesora new -g siuan@aessedai.com,moiraine@aessedai.com aessedai.gpg
```



This generates a new accounts file, `~/config/avendesora/aessedai.gpg`, and encrypts it so only you and Moiraine can open it. Mail this file to Moiraine. Since it is encrypted, it is safe to send the file through email. Even though Elaida can read this message, the accounts file is encrypted so she cannot access the master seed it contains. Moiraine should put the file in `~/config/avendesora` and then add it to `accounts_files` in `~/config/avendesora/accounts_files`. You are now ready to share accounts.

Then, when one partner creates a new account they mail the new account entry to the other partner. This entry does not contain enough information to allow an eavesdropper such as Elaida to be able to generate the secrets, but now both partners can. At a minimum you would need to share only the account name and the user name if one is needed. With that, the other partner can generate the passcode.

Once you have shared an accounts file, you can also use the *identity command* to prove your identity to your partner (described next).

You cannot share secrets encrypted with Scrypt. Also, you cannot share stealth accounts unless the file that contains the account templates has a *master\_seed* specified, which they do not by default. You would need to create a separate file for shared stealth account templates and add a master seed to that file manually.

### 3.5.10 Confirming the Identity of a Partner

The *identity command* allows you to generate a response to any challenge. The response identifies you to a remote partner with whom you have shared an account.

If you run the command with no arguments, it prints the list of valid names. If you run it with no challenge, one is created for you based on the current time and date.

If you have a remote partner to whom you wish to prove your identity, have that partner use *Avendesora* to generate a challenge and a response based on your shared secret. Then the remote partner provides you with the challenge and you run *Avendesora* with that challenge to generate the same response, which you provide to your remote partner to prove your identity.

You are free to explicitly specify a challenge to start the process, but it is important that it be unpredictable and that you not use the same challenge twice. As such, it is recommended that you not provide the challenge. In this situation, one is generated for you based on the time and date.

Consider an example that illustrates the process. In this example, Siuan is confirming the identity of Moiraine, where both Siuan and Moiraine are assumed to have shared *Avendesora* accounts. Siuan runs *Avendesora* as follows and remembers the response:

```
> avendesora identity moiraine
challenge: slouch emirate bedeck brooding
response: spear disable local marigold
```

This assumes that `moiraine` is the name, with any extension removed, of the file that Siuan uses to contain their shared accounts.

Siuan communicates the challenge to Moiraine but not the response. Moiraine then runs *Avendesora* with the given challenge:

```
> avendesora identity siuan slouch emirate bedeck brooding
challenge: slouch emirate bedeck brooding
response: spear disable local marigold
```

In this example, `siuan` is the name of the file that Moiraine uses to contain their shared accounts.

To complete the process, Moiraine returns the response to Siuan, who compares it to the response she received to confirm Moiraine's identity. If Siuan has forgotten the desired response, she can also specify the challenge to the *identity command* to regenerate the expected response.

Alternately, when Suan sends a message to Moiraine, she can proactively prove her identity by providing both the challenge and the response. Moiraine could then run the *identity* command with the challenge and confirm that she gets the same response. Other than herself, only Suan could predict the correct response to any challenge.

### 3.5.11 Phonetic Alphabet

When on the phone it can be difficult to convey the letters in an account identifier or other letter sequences. To help with this *Avendesora* can convert the sequence to the NATO phonetic alphabet. For example, imaging conveying the sequence '2WQI1T'. To do so, you can run the following:

```
> avendesora phonetic 2WQI1T
two whiskey quebec india one tango
```

Alternately, you can run the command without an argument, in which case it simply prints out the phonetic alphabet:

```
> avendesora p
Phonetic alphabet:
  Alfa      Echo      India    Mike     Quebec   Uniform  Yankee
  Bravo     Foxtrot   Juliett  November Romeo    Victor   Zulu
  Charlie   Golf      Kilo     Oscar    Sierra   Whiskey
  Delta     Hotel     Lima     Papa     Tango    X-ray
```

Now you can easily do the conversion yourself. Having *Avendesora* do the conversion for you helps you distinguish similar looking characters such as I and l and O and 0.

### 3.5.12 Upgrading from Abraxas

*Avendesora* generalizes and replaces *Abraxas*, its predecessor. To transition from *Abraxas* to *Avendesora*, you will first need to upgrade *Abraxas* to version 1.8 or higher (use 'abraxas -v' to determine version). Then run:

```
abraxas --export
```

It will create a collection of *Avendesora* accounts files in `~/.config/abraxas/avendesora`. You need to manually add these files to your list of accounts files in *Avendesora*. Say one such file is created: `~/.config/abraxas/avendesora/accounts.gpg`. This could be added to *Avendesora* as follows:

1. create a symbolic link from `~/.config/avendesora/abraxas_accounts.gpg` to `~/.config/abraxas/avendesora/accounts.gpg`:

```
cd ~/.config/avendesora
ln -s ../abraxas/avendesora/accounts.gpg abraxas_accounts.gpg
```

2. add `abraxas_accounts.gpg` to `account_files` list in `accounts_files`.

Now all of the *Abraxas* accounts contained in `abraxas_accounts.gpg` should be available though *Avendesora* and the various features of the account should operate as expected. However, secrets in accounts exported by *Abraxas* are no longer generated secrets. Instead, the actual secrets are placed in a hidden form in the exported accounts files.

If you would like to enhance the imported accounts to take advantage of the new features of *Avendesora*, it is recommended that you do not manually modify the imported files. Instead, copy the account information to one of your own account files before modifying it. To avoid conflict, you must then delete the account from the imported file. To do so, create `~/.config/abraxas/do-not-export` if it does not exist, then add the account name to this file, and reexport your accounts from *Abraxas*.

## 3.6 Command Reference

### 3.6.1 add – Add a new account

Usage:

```
avendesora add [options] [<template>]
avendesora a [options] [<template>]
```

Options:

<code>-f &lt;file&gt;, -file &lt;file&gt;</code>	Add account to specified accounts file.
--	---

Creates a new account starting from a template. The template consists of boilerplate code and fields. The fields take the form `_NAME_`. They should be replaced by appropriate values or deleted if not needed. If you are using the Vim editor, it is preconfigured to jump to the next field when you press ‘n’. If the field is surrounded by ‘<<’ and ‘>>’, as in ‘<<\_ACCOUNT\_NUMBER\_>>’, the value you enter will be concealed.

You can create your own templates by adding them to *account\_templates* in the `~/.config/avendesora/config` file.

You can change the editor used when adding account by changing the *edit\_template*, also found in the `~/.config/avendesora/config` file.

The default template is *bank*. The available templates are: *bank*, *shell*, and *website*.

### 3.6.2 archive – Generates archive of all account information

Usage:

```
avendesora archive
avendesora A
```

This command creates an encrypted archive that contains all the information in your accounts files, including the fully generated secrets. You should never need this file, but its presence protects you in case you lose access to Avendesora. To access your secrets without Avendesora, simply decrypt the archive file with GPG. The actual secrets will be hidden, but it is easy to retrieve them even without Avendesora. When hidden, the secrets are encoded in *base64*. You can decode it by running ‘base64 -d -’ and pasting the encoded secret into the terminal.

When you run this command it overwrites the existing archive. If you have accidentally deleted an account or changed a secret, then replacing the archive could cause the last copy of the original information to be lost. To prevent this from occurring it is a good practice to run the *changed* command before regenerating the archive. It describes all of the changes that have occurred since the last time the archive was generated. You should only regenerate the archive once you have convinced yourself all of the changes are as expected.

### 3.6.3 browse – Open account URL in web browser

Usage:

```
avendesora browse [options] <account> [<key>]
avendesora b [options] <account> [<key>]
```

Options:

-b <browser>, -browser <browser>	Open account in specified browser.
-l, -list	List available URLs rather than open first.

The account is examined for URLs, a URL is chosen, and then that URL is opened in the chosen browser. First URLs are gathered from the *urls* account attribute, which can be a string containing one or more URLs, a list, or a dictionary. If *urls* is a dictionary, the desired URL can be chosen by entering the key as an argument to the *browse command*. If a key is not given, then the *default\_url* account attribute is used to specify the key to use by default. If *urls* is not a dictionary, then the first URL specified is used.

If the *urls* attribute is not available then URLs are taken from *avendesora.RecognizeURL* objects in the *discovery* account attribute. In this case if the *name* argument is specified to *avendesora.RecognizeURL*, the corresponding URL can be chosen using a key.

The default browser is *x*, which uses the system default browser. You can override the default browser on a per-account basis by adding an attribute named *browser* to the account. An example of when you would specify the browser in an account would be an account associated with Tor hidden service, which generally can only be accessed using *torbrowser*:

```
class SilkRoad(Account):
    passcode = Passphrase()
    username = Passphrase(length=2, sep='-')
    url = 'http://silkroad6ownowfk.onion'
    browser = 't'
```

### 3.6.4 changed – Show changes since archive was created

Usage:

```
avendesora changed
avendesora C
```

When you run the *archive command* it overwrites the existing archive. If you have accidentally deleted an account or changed a secret, then replacing the archive could cause the last copy of the original information to be lost. To prevent this from occurring it is a good practice to run the *changed command* before regenerating the archive. It describes all of the changes that have occurred since the last time the archive was generated. You should only regenerate the archive once you have convinced yourself all of the changes are as expected.

### 3.6.5 conceal – Conceal text by encoding it

Usage:

```
avendesora conceal [options] [<text>]
avendesora c      [options] [<text>]
```

Options:

<code>-e &lt;encoding&gt;, -encoding &lt;encoding&gt;</code>	Encoding used when concealing information.
<code>-g &lt;id&gt;, -gpg-id &lt;id&gt;</code>	Use this ID when creating any missing encrypted files. Use commas with no spaces to separate multiple IDs.
<code>-h &lt;path&gt;, -gpg-home &lt;path&gt;</code>	GPG home directory (default is <code>~/gnupg</code> ).
<code>-s, -symmetric</code>	Encrypt with a passphrase rather than using your GPG key (only appropriate for gpg encodings).

Possible encodings include (default encoding is base64):

**gpg:** This encoding fully encrypts/decrypts the text with GPG key. By default your GPG key is used, but you can specify symmetric encryption, in which case a passphrase is used.

**base64:** This encoding obscures but does not encrypt the text. It can protect text from observers that get a quick glance of the encoded text, but if they are able to capture it they can easily decode it.

**script:** This encoding fully encrypts the text with your user key. Only you can decrypt it, secrets encoded with script cannot be shared.

Though available as an option for convenience, you should not pass the text to be hidden as an argument as it is possible for others to examine the commands you run and their argument list. For any sensitive secret, you should simply run ‘avendesora conceal’ and then enter the secret text when prompted.

### 3.6.6 credentials – Show login credentials

Displays the account’s login credentials, which generally consist of an identifier and a secret.

Usage:

```
avendesora credentials [options] <account>
avendesora login      [options] <account>
avendesora l          [options] <account>
```

Options:

<code>-S, -seed</code>	Interactively request additional seed for generated secrets.
------------------------	--

The credentials can be specified explicitly using the credentials setting in your account. For example:

```
credentials = 'usernames.0 usernames.1 passcode'
```

If credentials is not specified then the first of the following will be used if available:

id: username or email

secret: passcode, password or passphrase

### 3.6.7 edit – Edit an account

Usage:

```
avendesora edit <account>
avendesora e    <account>
```

Opens an existing account in your editor.

You can specify the editor by changing the *edit\_account* setting in the config file (*~/config/avendesora/config*).

### 3.6.8 find – Find an account

Find accounts whose name contains the search text.

Usage:

```
avendesora find <text>
avendesora f    <text>
```

### 3.6.9 help – Give information about commands or other topics

Usage:

```
avendesora help [options] [<topic>]
avendesora h    [options] [<topic>]
```

Options:

<code>-s, --search</code>	list topics that include <code>&lt;topic&gt;</code> as a search term.
<code>-b, --browse</code>	open the topic in your default browser.

You can also use `avendesora --help` or `avendesora -h` to see the global options for *Avendesora*.

### 3.6.10 identity – Generate an identifying response to a challenge

Usage:

```
avendesora identity [<name> [<challenge>...]]
avendesora ident    [<name> [<challenge>...]]
avendesora i        [<name> [<challenge>...]]
```

This command allows you to generate a response to any challenge. The response identifies you to a partner with whom you have shared an account.

If you run the command with no arguments, it prints the list of available accounts. If you run it with no challenge, one is created for you based on the current time and date.

If you have a remote partner to whom you wish to prove your identity, have that partner use *avendesora* to generate a challenge and a response based on your shared secret. Then the remote partner provides you with the challenge and you run *avendesora* with that challenge to generate the same response, which you provide to your remote partner to prove your identity.

You are free to explicitly specify a challenge to start the process, but it is important that it be unpredictable and that you not use the same challenge twice. As such, it is recommended that you not provide the challenge. In this situation, one is generated for you based on the time and date.

See *Confirming the Identity of a Partner* for an example that illustrates the process.

### 3.6.11 initialize – Create initial set of Avendesora files

Usage:

```
avendesora initialize [options]
avendesora init      [options]
avendesora I         [options]
```

Options:

-g <id>, -gpg-id <id>	Use this ID when creating any missing encrypted files. Use commas with no spaces to separate multiple IDs.
-h <path>, -gpg-home <path>	GPG home directory (default is ~/.gnupg).

Create Avendesora data directory (~/.config/avendesora) and populate it with initial versions of all essential files.

It is safe to run this command even after the data directory and files have been created. Doing so will simply recreate any missing files. Existing files are not modified.

### 3.6.12 log – Open the logfile

Usage:

```
avendesora log
```

Opens the logfile in your editor.

You can specify the editor by changing the *edit\_account* setting in the config file (~/.config/avendesora/config).

### 3.6.13 new – Create new accounts file

Usage:

```
avendesora new [options] <name>
avendesora N   [options] <name>
```

Options:

-g <id>, -gpg-id <id>	Use this ID when creating any missing encrypted files. Use commas with no spaces to separate multiple IDs.
-----------------------	--

Creates a new accounts file. Accounts that share the same file share the same master seed by default and, if the file is encrypted, can be decrypted by the same recipients.

Generally you create new accounts files for each person or group with which you wish to share accounts. You also use separate files for passwords with different security domains. For example, a high-value passwords might be placed in an encrypted file that would only be placed highly on protected computers. Conversely, low-value passwords might be contained in perhaps an unencrypted file that is found on many computers.

Add a '.gpg' extension to <name> to encrypt the file.

### 3.6.14 phonetic – Display NATO phonetic alphabet

Usage:

```
avendesora alphabet [<text>]
avendesora phonetic [<text>]
avendesora p [<text>]
```

If <text> is given, any letters are converted to the phonetic alphabet. If not given the entire phonetic is displayed.

Example:

```
> avendesora phonetic 2WQI1T
two whiskey quebec india one tango

> avendesora phonetic
Phonetic alphabet:
  Alfa      Echo      India    Mike     Quebec   Uniform  Yankee
  Bravo     Foxtrot  Juliett  November Romeo    Victor   Zulu
  Charlie   Golf     Kilo     Oscar    Sierra   Whiskey
  Delta     Hotel    Lima     Papa     Tango    X-ray
```

### 3.6.15 reveal – Reveal concealed text

Transform concealed text to reveal its original form.

Usage:

```
avendesora reveal [<text>]
avendesora r      [<text>]
```

**Options:**

<code>-e &lt;encoding&gt;, -encoding &lt;encoding&gt;</code>	Encoding used when revealing information.
--	---

Though available as an option for convenience, you should not pass the text to be revealed as an argument as it is possible for others to examine the commands you run and their argument list. For any sensitive secret, you should simply run ‘avendesora reveal’ and then enter the encoded text when prompted.

### 3.6.16 search – Search accounts

Search for accounts whose values contain the search text.

Usage:

```
avendesora search <text>
avendesora s      <text>
```

### 3.6.17 value – Show an account value

Produce an account value. If the value is secret, it is produced only temporarily unless `--stdout` is specified.

Usage:



```
avendesora value [options] [<account> [<field>]]
avendesora val  [options] [<account> [<field>]]
avendesora v    [options] [<account> [<field>]]
```

**Options:**

-c, -clipboard	Write output to clipboard rather than stdout.
-s, -stdout	Write output to the standard output without any annotation or protections.
-S, -seed	Interactively request additional seed for generated secrets.
-v, -verbose	Add additional information to log file to help identify issues in account discovery.
-T <title>, -title <title>	Use account discovery on this title.

The ‘vc’ command is a shortcut for ‘value -clipboard’.

You request a scalar value by specifying its name after the account. For example:

```
avendesora value bank pin
```

If the requested value is composite (an array or dictionary), you should also specify a key that indicates which of the composite values you want. For example, if the *accounts* field is a dictionary, you specify *accounts.checking* or *accounts[checking]* to get information on your checking account. If the value is an array, you give the index of the desired value. For example, *questions.0* or *questions[0]*. If you only specify a number, then the name is assumed to be *questions*, as in the list of security questions (this can be changed by specifying the desired name as the *default\_vector\_field* setting).

The field may be also be a script, with is nothing but a string that it output as given except that embedded attributes are replaced by account field values. For example:

```
avendesora value bank '{accounts.checking}: {passcode}'
```

If no value is requested the result produced is determined by the value of the *default* attribute. If no value is given for *default*, then the *passcode*, *password*, or *passphrase* attribute is produced (this can be changed by specifying the *default\_field* setting). If *default* is a *script* then the script is executed. A typical script might be ‘username: {username}, password: {passcode}’. It is best if the script produces a one line output if it contains secrets. If not a script, the value of *default* should be the name of another attribute, and the value of that attribute is shown.

If no account is requested, then *Avendesora* attempts to determine the appropriate account through *discovery*. Normally *Avendesora* is called in this manner from your window manager. You would arrange for it to be run when you type a hot key. In this case *Avendesora* determines which account to use from information available from the environment, information like the title on active window. In this mode, *Avendesora* mimics the keyboard when producing its output.

The *verbose* and *title* options are used when debugging account discovery. The *verbose* option adds more information about the discovery process to the logfile (*~/config/avendesora/log.gpg*). The *title* option allows you to override the active window title so you can debug title-based discovery. Specifying the *title* option also scrubs the output and outputs directly to the standard output rather than mimicking the keyboard so as to avoid exposing your secret.

If the *stdout* option is not specified, the value command still writes to the standard output if it is associated with a TTY (if *Avendesora* is outputting directly to a terminal). If the standard output is not a TTY, *Avendesora* mimics the keyboard and types the desired value directly into the active window. There are two common situations where standard output is not a TTY: when *Avendesora* is being run by your window manager in response to you pressing a hot key or when the output of *Avendesora* is fed into a pipeline. In the second case, mimicking the keyboard is not what you want; you should use *--stdout* to assure the chosen value is sent to the pipeline as desired. This also has the added benefit of stripping off all decorations from the value.

### 3.6.18 values – Display all account values

Show all account values.

Usage:

```
avendesora values <account>
avendesora vals   <account>
avendesora V      <account>
```

The values of secrets are not actually shown. Rather instructions for viewing the secret value is given. Also, account attributes that are intended only to control *Avendesora*, such as *discovery*, are not shown at all.

### 3.6.19 version – Display Avendesora version

Usage:

```
avendesora version
```

## 3.7 Account Helpers

Account helpers are classes and character sets that are used in your account attributes. They provide advanced capabilities such as holding secrets, generating secrets, and recognizing accounts.

### 3.7.1 Generated Secret Classes

Subclasses of `avendesora.GeneratedSecret`.

These classes are used when creating account secrets (see *Accounts*).

Every class starts with a pool of 512 bits of entropy. Each symbol generated consumes some of that entropy, the amount of which is determined by the number of symbols that are available in the alphabet. For example, passphrases pull words from a dictionary containing 10,000 words. As such, each word in the passphrase consumes 14 bits of entropy ( $\text{ceil}(\log_2(10000))$ ). If too many words are requested, `avendesora.SecretExhausted` is raised.

```
class avendesora.Password (length=12, alphabet='abcdefghijklmnopqrstuvwxyABCDEFGHIJKLMNOPQRSTUVWXYZ234567890!@#$%^&*~`-+=_&#%#@',
                           master=None, version=None, shift_sort=False, sep="", prefix="", suffix="", is_secret=True)
```

Generate password.

Generates an arbitrary password by selecting symbols from the given alphabet at random. The entropy of the generated password is  $\text{length} \times \log_2(\text{len}(\text{alphabet}))$ .

#### Parameters

- **length** (*int*) – The number of items to draw from the alphabet when creating the password.
- **alphabet** (*collection of symbols*) – The reservoir of legal symbols to use when creating the password. By default the set of easily distinguished alphanumeric characters are used (`avendesora.DISTINGUISHABLE`). Typically you would use the pre-imported character sets to construct the alphabet. For example, you might pass: `avendesora.ALPHANUMERIC + '+=_&%#@'`

- **master** (*str*) – Overrides the master seed that is used when generating the password. Generally, there is one master seed shared by all accounts contained in an account file. This argument overrides that behavior and instead explicitly specifies the master seed for this secret.
- **version** (*str*) – An optional seed. Changing this value will change the generated password.
- **shift\_sort** (*bool*) – If true, the characters in the password will be sorted so that the characters that require the shift key when typing are placed last. This make the password easier to type.
- **sep** (*str*) – A string that is placed between each symbol in the generated password.
- **prefix** (*str*) – A string added to the front of the generated password.
- **suffix** (*str*) – A string added to the end of the generated password.
- **is\_secret** (*bool*) – Should value be hidden from user unless explicitly requested.

**Raises** `avendesora.SecretExhausted` – The available entropy has been exhausted. This occurs when the requested length is too long.

Examples:

```
>>> secret = Password()
>>> secret.initialize(account, 'dux')
>>> str(secret)
'tvA8mewbbig3'

>>> secret = Password(shift_sort=True)
>>> secret.initialize(account, 'flux')
>>> str(secret)
'wrncpipvtNPF'
```

**class** `avendesora.Passphrase` (*length=4, alphabet=None, master=None, version=None, sep=' ', prefix=", suffix=", is\_secret=True*)

Generate passphrase.

Similar to `Password` in that it generates an arbitrary passphrase by selecting symbols from the given alphabet at random, but in this case the default alphabet is a dictionary containing about 10,000 words.

#### Parameters

- **length** (*int*) – The number of items to draw from the alphabet when creating the password.
- **alphabet** (*collection of symbols*) – The reservoir of legal symbols to use when creating the password. By default this is a list of 10,000 words.
- **master** (*str*) – Overrides the master seed that is used when generating the password. Generally, there is one master seed shared by all accounts contained in an account file. This argument overrides that behavior and instead explicitly specifies the master seed for this secret.
- **version** (*str*) – An optional seed. Changing this value will change the generated password.
- **sep** (*str*) – A string that is placed between each symbol in the generated password.
- **prefix** (*str*) – A string added to the front of the generated password.
- **suffix** (*str*) – A string added to the end of the generated password.

- **is\_secret** (*bool*) – Should value be hidden from user unless explicitly requested.

**Raises** *avendesora.SecretExhausted* – The available entropy has been exhausted. This occurs when the requested length is too long.

Example:

```
>>> secret = Passphrase()
>>> secret.initialize(account, 'dux')
>>> str(secret)
'graveyard cockle intone provider'
```

**class** *avendesora.PIN* (*length=4, alphabet='0123456789', master=None, version=None, is\_secret=True*)

Generate PIN.

Similar to *Passphrase* in that it generates an arbitrary PIN by selecting symbols from the given alphabet at random, but in this case the default alphabet is the set of digits (0-9).

#### Parameters

- **length** (*int*) – The number of items to draw from the alphabet when creating the password.
- **alphabet** (*collection of symbols*) – The reservoir of legal symbols to use when creating the password. By default the alphabet is *avendesora.DIGITS*.
- **master** (*str*) – Overrides the master seed that is used when generating the password. Generally, there is one master seed shared by all accounts contained in an account file. This argument overrides that behavior and instead explicitly specifies the master seed for this secret.
- **version** (*str*) – An optional seed. Changing this value will change the generated password.
- **sep** (*str*) – A string that is placed between each symbol in the generated password.
- **prefix** (*str*) – A string added to the front of the generated password.
- **suffix** (*str*) – A string added to the end of the generated password.
- **is\_secret** (*bool*) – Should value be hidden from user unless explicitly requested.

**Raises** *avendesora.SecretExhausted* – The available entropy has been exhausted. This occurs when the requested length is too long.

Example:

```
>>> secret = PIN()
>>> secret.initialize(account, 'dux')
>>> str(secret)
'9301'
```

**class** *avendesora.Question* (*question, length=3, alphabet=None, master=None, version=None, sep=' ', prefix="", suffix="", answer=None, is\_secret=True*)

Generate arbitrary answer to a given question.

Similar to *Passphrase()* except a question must be specified when created and it is taken to be the security question. The question is used as a seed rather than the field name when generating the secret.

#### Parameters

- **question** (*str*) – The question to be answered. Be careful. Changing the question in any way will change the resulting answer.

- **length** (*int*) – The number of items to draw from the alphabet when creating the answer.
- **alphabet** (*collection of symbols*) – The reservoir of legal symbols to use when creating the password.
- **master** (*str*) – Overrides the master seed that is used when generating the password. Generally, there is one master seed shared by all accounts contained in an account file. This argument overrides that behavior and instead explicitly specifies the master seed for this secret.
- **version** (*str*) – An optional seed. Changing this value will change the generated password.
- **sep** (*str*) – A string that is placed between each symbol in the generated password.
- **prefix** (*str*) – A string added to the front of the generated password.
- **suffix** (*str*) – A string added to the end of the generated password.
- **answer** (*str*) – The answer. If provided, this would override the generated answer. May be a string, or it may be an Obscured object.
- **is\_secret** (*bool*) – Should value be hidden from user unless explicitly requested.

**Raises** *avendesora.SecretExhausted* – The available entropy has been exhausted. This occurs when the requested length is too long.

### Example

```
>>> secret = Question('What city were you born in?')
>>> secret.initialize(account, 'dux')
>>> str(secret)
'dustcart olive label'
```

**class** *avendesora.MixedPassword* (*length, def\_alphabet, requirements, master=None, version=None, shift\_sort=False, is\_secret=True*)

Generate mixed password.

A relatively low level method that is used to generate passwords from a heterogeneous collection of alphabets. This is used to satisfy the character type count requirements of many websites. It is recommended that user use *avendesora.PasswordRecipe* rather than directly use this class.

#### Parameters

- **length** (*int*) – The number of items to draw from the various alphabets when creating the password.
- **def\_alphabet** (*collection of symbols*) – The alphabet to use when filling up the password after all the constraints are satisfied.
- **requirements** (*list of tuples*) – Each tuple has two members, the first is a string or list that is used as an alphabet, and the second is a number that indicates how many symbols should be drawn from that alphabet.
- **master** (*str*) – Overrides the master seed that is used when generating the password. Generally, there is one master seed shared by all accounts contained in an account file. This argument overrides that behavior and instead explicitly specifies the master seed for this secret.
- **version** (*str*) – An optional seed. Changing this value will change the generated answer.

- **shift\_sort** (*bool*) – If true, the characters in the password will be sorted so that the characters that require the shift key when typing are placed last. This make the password easier to type.
- **is\_secret** (*bool*) – Should value be hidden from user unless explicitly requested.

**Raises** *avendesora.SecretExhausted* – The available entropy has been exhausted. This occurs when the requested length is too long.

Example:

```
>>> secret = MixedPassword(
...     12, ALPHANUMERIC, [(LOWERCASE, 2), (UPPERCASE, 2), (DIGITS, 2)]
... )
>>> secret.initialize(account, 'dux')
>>> str(secret)
'ZyW62FvxX0Fg'
```

**class** *avendesora.PasswordRecipe* (*recipe*, *def\_alphabet*='abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZMNOPQRSTUVWXYZ', *master*=None, *version*=None, *shift\_sort*=False, *is\_secret*=True)

Generate password from recipe.

A version of *MixedPassword* where the requirements are specified with a short string rather than using the more flexible but more cumbersome method of *MixedPassword*. The string consists of a series of terms separated by white space. The first term is a number that specifies the total number of characters in the password. The remaining terms specify the number of characters that should be pulled from a particular class of characters. The classes are u (upper case letters), l (lower case letters), d (digits), s (punctuation), and c (an explicitly specified set of characters). For example, '12 2u 2d 2s' indicates that a 12 character password should be generated that includes 2 upper case letters, 2 digits, and 2 symbols. The remaining characters will be chosen from the base character set, which by default is the set of alphanumeric characters.

#### Parameters

- **recipe** (*str*) – A string that describes how the password should be constructed.
- **def\_alphabet** (*collection of symbols*) – The alphabet to use when filling up the password after all the constraints are satisfied.
- **master** (*str*) – Overrides the master seed that is used when generating the password. Generally, there is one master seed shared by all accounts contained in an account file. This argument overrides that behavior and instead explicitly specifies the master seed for this secret.
- **version** (*str*) – An optional seed. Changing this value will change the generated answer.
- **shift\_sort** (*bool*) – If true, the characters in the password will be sorted so that the characters that require the shift key when typing are placed last. This make the password easier to type.
- **is\_secret** (*bool*) – Should value be hidden from user unless explicitly requested.

**Raises** *avendesora.SecretExhausted* – The available entropy has been exhausted. This occurs when the requested length is too long.

Example:

```
>>> secret = PasswordRecipe('12 2u 2d 2s')
>>> secret.initialize(account, 'pux')
>>> str(secret)
'*m7Aqj=XBAs7'
```

The `c` class is special in that it allow you to explicitly specify the characters to use. For example, `'12 2c!@#%^^&='` directs that a 12 character password be generated, 2 of which are taken from the set `!@#%^^&=`:

```
>>> secret = PasswordRecipe('12 2u 2d 2c!@#%^^&*')
>>> secret.initialize(account, 'bux')
>>> str(secret)
'Y08K^68J9oC!'
```

**class** `avendesora.BirthDate` (*year*, *min\_age=18*, *max\_age=65*, *fmt='YYYY-MM-DD'*, *master=None*, *version=None*, *is\_secret=True*)

Generates an arbitrary birthdate for someone in a specified age range.

This function can be used to generate an arbitrary date using:

```
>>> secret = BirthDate(2015, 18, 65)
>>> secret.initialize(account, 'dux')
>>> str(secret)
'1970-03-22'
```

For *year*, enter the year the account that contains `BirthDate` was created. Doing so anchors the age range. In this example, the creation date is 2015, the minimum age is 18 and the maximum age is 65, meaning that a birthdate will be chosen such that in 2015 the birth date could correspond to someone that is between 18 and 65 years old.

You can use the *fmt* argument to change the way in which the date is formatted:

```
>>> secret = BirthDate(2015, 18, 65, fmt="M/D/YY")
>>> secret.initialize(account, 'dux')
>>> str(secret)
'3/22/70'
```

### Parameters

- **year** (*int*) – The year the age range was established.
- **min\_age** (*int*) – The lower bound of the age range.
- **max\_age** (*int*) – The upper bound of the age range.
- **fmt** (*str*) – Specifies the way the date is formatted. Consider an example date of 6 July 1969. `YY` and `YYYY` are replaced by the year (69 or 1969). `M`, `MM`, `MMM`, and `MMMM` are replaced by the month (7, 07, Jul, or July). `D` and `DD` are replaced by the day (6 or 06).
- **master** (*str*) – Overrides the master seed that is used when generating the password. Generally, there is one master seed shared by all accounts contained in an account file. This argument overrides that behavior and instead explicitly specifies the master seed for this secret.
- **version** (*str*) – An optional seed. Changing this value will change the generated answer.
- **is\_secret** (*bool*) – Should value be hidden from user unless explicitly requested.

**Raises** `avendesora.SecretExhausted` – The available entropy has been exhausted. This occurs when the requested length is too long.

**class** `avendesora.OTP` (*shared\_secret*, *interval=30*, *digits=6*)

One Time Password

Generates a secret that changes over time that generally is used as a second factor when authenticating. It can act as a replacement for, and is fully compatible with, Google Authenticator or Authy. You would provide the text version of the shared secret that is presented to you when first configuring your second factor authentication. See *One-Time Passwords* for more information.

Only available if pyotp is installed (pip install pyotp).

**Parameters**

- **shared\_secret** (*str*) – The shared secret in base32.
- **interval** (*int*) – Update interval in seconds. Use 30 to mimic Google Authenticator, 10 to mimic Authy.
- **digits** (*int*) – Number of digits to output, choose between 6, 7, or 8. Use 6 to mimic Google Authenticator, 7 to mimic Authy.

**exception** `avendesora.SecretExhausted` (\*\*kwargs)  
 Secret exhausted.

This generally results if the length of the requested secret is too long.

This exception subclasses `avendesora.PasswordError`.

**report** (*template=None*)  
 Report exception.

The `inform.error()` function is called with the exception arguments.

**Parameters** **template** (*str*) – This argument is treated as a format string and is passed both the unnamed and named arguments. The resulting string is treated as the message and returned.

If not specified, the *template* keyword argument passed to the exception is used. If there was no *template* argument, then the positional arguments of the exception are joined using *sep* and that is returned.

**terminate** (*template=None*)  
 Report exception and terminate.

The `inform.fatal()` function is called with the exception arguments.

**Parameters** **template** (*str*) – This argument is treated as a format string and is passed both the unnamed and named arguments. The resulting string is treated as the message and returned.

If not specified, the *template* keyword argument passed to the exception is used. If there was no *template* argument, then the positional arguments of the exception are joined using *sep* and that is returned.

### 3.7.2 Character Sets

These are useful when constructing generated secrets. They are used to build the alphabet used by the generator. For example, you can specify that passwords should be constructed from 12 lower case letters and digits with:

```
Password(length=12, alphabet=LOWERCASE+DIGITS)
```

Or here is an example that starts with the alphanumeric and punctuation characters, and removes those that require the shift key to type:

```
Password(length=12, alphabet=exclude(ALPHANUMERIC+PUNCTUATION, SHIFTED))
```

`avendesora.exclude` (*chars, exclusions*)  
 Exclude Characters

Use this to remove characters from a character set.



**Parameters**

- **chars** (*str*) – Character set to strip.
- **exclusions** (*str*) – Characters to remove from character set.

Example:

```
>>> exclude('ABCDEF', 'AEF')
'BCD'
```

**avendesora.LOWERCASE**

Lower case ASCII letters: *avendesora.LOWERCASE* = “abcdefghijklmnopqrstuvwxyz”

**avendesora.UPPERCASE**

Upper case ASCII letters: *avendesora.UPPERCASE* = “ABCDEFGHIJKLMNOPQRSTUVWXYZ”

**avendesora.LETTERS**

Upper and lower case ASCII letters: *avendesora.LETTERS* = *avendesora.LOWERCASE* + *avendesora.UPPERCASE*

**avendesora.DIGITS**

ASCII digits: *avendesora.DIGITS* = “0123456789”

**avendesora.ALPHANUMERIC**

ASCII letters and digits: *avendesora.ALPHANUMERIC* = *avendesora.LETTERS* + *avendesora.DIGITS*

**avendesora.HEXDIGITS**

Hexidecimal digits: *avendesora.HEXDIGITS* = “0123456789abcdef”

**avendesora.PUNCTUATION**

ASCII punctuation characters: *avendesora.PUNCTUATION* = “!”#\$%&'()\*+,-./:;<=>?@[\\]^\_`{|}~”

**avendesora.SYMBOLS**

ASCII punctuation characters excluding ‘, ‘, ‘, and \: *avendesora.SYMBOLS* = `exclude(avendesora.PUNCTUATION, “‘‘‘”)`

**avendesora.WHITESPACE**

ASCII white space characters (excluding newlines): *avendesora.WHITESPACE* = “\t”

**avendesora.PRINTABLE**

All ASCII printable characters (letters, digits, punctuation, whitespace): *avendesora.PRINTABLE* = *avendesora.ALPHANUMERIC* + *avendesora.PUNCTUATION* + *avendesora.WHITESPACE*

**avendesora.DISTINGUISHABLE**

ASCII letters and digits with easily confused characters removed: *avendesora.DISTINGUISHABLE* = `exclude(avendesora.ALPHANUMERIC, ‘II1O0’)`

**avendesora.SHIFTED**

ASCII characters that are typed using the shift key: *avendesora.SHIFTED* = *avendesora.UPPERCASE* + “~!@#\$%^&\*()\_+{|};’<>?”

### 3.7.3 Obscured Secret Classes

Sublasses of *avendesora.ObscuredSecret*.

These classes are used when creating account secrets (see *Accounts*).

```
class avendesora.Hide (plaintext, secure=True, is_secret=True)
    Hide text
```

Marks a value as being secret.

#### Parameters

- **plaintext** (*str*) – The value of interest.
- **secure** (*bool*) – Indicates that this secret is of high value and so should not be found in an unencrypted accounts file.
- **is\_secret** (*bool*) – Should value be hidden from user unless explicitly requested.

**class** `avendesora.Hidden` (*encoded\_text*, *secure=True*, *encoding=None*, *is\_secret=True*)

Hidden text

This encoding obscures but does not encrypt the text.

#### Parameters

- **encoded\_text** (*str*) – The value of interest encoded in base64.
- **secure** (*bool*) – Indicates that this secret is of high value and so should not be found in an unencrypted accounts file.
- **encoding** (*str*) – The encoding to use for the decoded text.
- **is\_secret** (*bool*) – Should value be hidden from user unless explicitly requested.

**Raises** `avendesora.PasswordError` – invalid value.

**class** `avendesora.GPG` (*ciphertext*, *secure=True*, *encoding=None*)

GPG encrypted text

The secret is fully encrypted with GPG. Both symmetric encryption and key-based encryption are supported.

#### Parameters

- **ciphertext** (*str*) – The secret encrypted and armored by GPG.
- **encoding** (*str*) – The encoding to use for the deciphered text.

**Raises** `avendesora.PasswordError` – invalid value.

## 3.7.4 Recognizer Classes

These classes are used in *account discovery*.

**class** `avendesora.RecognizeAll` (*\*recognizers*, *\*\*kwargs*)

Run script if all recognizers match.

Takes one or more recognizers. Script is run if all recognizers match.

#### Parameters

- **recognizer** (*Recognizer*) – One or more instances of `Recognizer`.
- **script** (*str* or *True*) – A script that indicates the text that should be typed to active application. The names of fields can be included in the script surrounded by braces, in which case the value of the field replaces the field reference. For example:

```
Script('username: {username}, password: {passcode}')
```

In this case, *{username}* and *{passcode}* are replaced by with the value of the corresponding account attribute. In addition to the fields, *{tab}* and *{return}* are replaced by a tab or carriage return character, and *{sleep N}* causes the typing to pause for *N* seconds.

If True is give, the default field is produced followed by a return.

**Raises** *avendesora.PasswordError*

**class** *avendesora.RecognizeAny* (\*recognizers, \*\*kwargs)

Run script if any recognizers match.

Takes one or more recognizers. Script is run if any recognizers match.

#### Parameters

- **recognizer** (*Recognizer*) – One or more instances of *Recognizer*.
- **script** (*str* or *True*) – A script that indicates the text that should be typed to active application. The names of fields can be included in the script surrounded by braces, in which case the value of the field replaces the field reference. For example:

```
Script('username: {username}, password: {passcode}')
```

In this case, *{username}* and *{passcode}* are replaced by with the value of the corresponding account attribute. In addition to the fields, *{tab}* and *{return}* are replaced by a tab or carriage return character, and *{sleep N}* causes the typing to pause for *N* seconds.

If True is give, the default field is produced followed by a return.

**Raises** *avendesora.PasswordError*

**class** *avendesora.RecognizeTitle* (\*titles, \*\*kwargs)

Run script if window title matches.

Takes one or more glob strings. Script is run if window title matches any of the glob strings.

#### Parameters

- **title** (*str*) – One or more glob strings.
- **script** (*str* or *True*) – A script that indicates the text that should be typed to active application. The names of fields can be included in the script surrounded by braces, in which case the value of the field replaces the field reference. For example:

```
Script('username: {username}, password: {passcode}')
```

In this case, *{username}* and *{passcode}* are replaced by with the value of the corresponding account attribute. In addition to the fields, *{tab}* and *{return}* are replaced by a tab or carriage return character, and *{sleep N}* causes the typing to pause for *N* seconds.

If True is give, the default field is produced followed by a return.

**Raises** *avendesora.PasswordError*

**class** *avendesora.RecognizeURL* (\*urls, \*\*kwargs)

Run script if URL matches.

Takes one or more URLs. Script is run if URL embedded in window title matches any of the given URLs. Assumes that a browser plugin has embedded the URL in the browser's window title. This is generally safer and more robust than *RecognizeTitle* when trying to match web pages.

When giving the URL, anything specified must match and globbing is not supported. If you give a partial path, by default *Avendesora* will match up to what you have given, but you can require an exact match of the entire path by specifying `exact_path=True` to *RecognizeURL*. If you do not give the protocol, the `default_protocol` (`https`) is assumed.

#### Parameters

- **url** (*str*) – One or more URLs.
- **exact\_path** (*bool*) – If True, path given in the URL must be matched completely, partial matches are ignored.
- **script** (*str or True*) – A script that indicates the text that should be typed to active application. The names of fields can be included in the script surrounded by braces, in which case the value of the field replaces the field reference. For example:

```
Script('username: {username}, password: {passcode}')
```

In this case, *{username}* and *{passcode}* are replaced by with the value of the corresponding account attribute. In addition to the fields, *{tab}* and *{return}* are replaced by a tab or carriage return character, and *{sleep N}* causes the typing to pause for *N* seconds.

If True is give, the default field is produced followed by a return.

**Raises** *avendesora.PasswordError*

**class** *avendesora.RecognizeCWD* (*\*dirs, \*\*kwargs*)

Run script if current working directory matches.

Takes one or more paths. Script is run if any path refers to the current working directory.

#### Parameters

- **path** (*str*) – One or more directory paths.
- **script** (*str or True*) – A script that indicates the text that should be typed to active application. The names of fields can be included in the script surrounded by braces, in which case the value of the field replaces the field reference. For example:

```
Script('username: {username}, password: {passcode}')
```

In this case, *{username}* and *{passcode}* are replaced by with the value of the corresponding account attribute. In addition to the fields, *{tab}* and *{return}* are replaced by a tab or carriage return character, and *{sleep N}* causes the typing to pause for *N* seconds.

If True is give, the default field is produced followed by a return.

**Raises** *avendesora.PasswordError*

**class** *avendesora.RecognizeHost* (*\*hosts, \*\*kwargs*)

Run script if host name matches.

Takes one or more host names. Script is run if the current host name matches one of the given host names.

#### Parameters

- **host** (*str*) – One or more host names.
- **script** (*str or True*) – A script that indicates the text that should be typed to active application. The names of fields can be included in the script surrounded by braces, in which case the value of the field replaces the field reference. For example:

```
Script('username: {username}, password: {passcode}')
```

In this case, *{username}* and *{passcode}* are replaced by with the value of the corresponding account attribute. In addition to the fields, *{tab}* and *{return}* are replaced by a tab or carriage return character, and *{sleep N}* causes the typing to pause for *N* seconds.

If True is give, the default field is produced followed by a return.

**Raises** *avendesora.PasswordError*

**class** avendesora.**RecognizeUser** (\*users, \*\*kwargs)

Run script if user name matches.

Takes one or more user names. Script is run if the current user name matches one of the given user names.

#### Parameters

- **user** (*str*) – One or more user names.
- **script** (*str* or *True*) – A script that indicates the text that should be typed to active application. The names of fields can be included in the script surrounded by braces, in which case the value of the field replaces the field reference. For example:

```
Script('username: {username}, password: {passcode}')
```

In this case, *{username}* and *{passcode}* are replaced by with the value of the corresponding account attribute. In addition to the fields, *{tab}* and *{return}* are replaced by a tab or carriage return character, and *{sleep N}* causes the typing to pause for *N* seconds.

If *True* is give, the default field is produced followed by a return.

**Raises** *avendesora.PasswordError*

**class** avendesora.**RecognizeEnvVar** (name, value, script=*True*)

Run script if environment variable matches.

Script is run if the environment variable exists and its value matches the value given.

#### Parameters

- **name** (*str*) – Name of environment variable.
- **value** (*str*) – Value of environment variable.
- **script** (*str* or *True*) – A script that indicates the text that should be typed to active application. The names of fields can be included in the script surrounded by braces, in which case the value of the field replaces the field reference. For example:

```
Script('username: {username}, password: {passcode}')
```

In this case, *{username}* and *{passcode}* are replaced by with the value of the corresponding account attribute. In addition to the fields, *{tab}* and *{return}* are replaced by a tab or carriage return character, and *{sleep N}* causes the typing to pause for *N* seconds.

If *True* is give, the default field is produced followed by a return.

**Raises** *avendesora.PasswordError*

**class** avendesora.**RecognizeNetwork** (\*macs, \*\*kwargs)

Recognize network from MAC address.

Matches if any of the MAC addresses reported by */sbin/arp* match any of those given as an argument.

#### Parameters

- **mac** (*str*) – MAC address given in the form: '00:c9:a9:f7:30:00'.
- **script** (*str* or *True*) – A script that indicates the text that should be typed to active application. The names of fields can be included in the script surrounded by braces, in which case the value of the field replaces the field reference. For example:

```
Script('username: {username}, password: {passcode}')
```

In this case, `{username}` and `{passcode}` are replaced by with the value of the corresponding account attribute. In addition to the fields, `{tab}` and `{return}` are replaced by a tab or carriage return character, and `{sleep N}` causes the typing to pause for  $N$  seconds.

If `True` is give, the default field is produced followed by a return.

**Raises** `avendesora.PasswordError`

**class** `avendesora.RecognizeFile` (`filepath`, `contents=None`, `wait=60`, `**kwargs`)  
Recognize file.

Matches if file exists and was created within the last few seconds.

#### Parameters

- **filepath** (`str`) – Path to file.
- **contents** (`str`) – Expected file contents. If given, should match contents of file.
- **wait** (`float`) – Do not match if file is older than this value (seconds).
- **script** (`str` or `True`) – A script that indicates the text that should be typed to active application. The names of fields can be included in the script surrounded by braces, in which case the value of the field replaces the field reference. For example:

```
Script('username: {username}, password: {passcode}')
```

In this case, `{username}` and `{passcode}` are replaced by with the value of the corresponding account attribute. In addition to the fields, `{tab}` and `{return}` are replaced by a tab or carriage return character, and `{sleep N}` causes the typing to pause for  $N$  seconds.

If `True` is give, the default field is produced followed by a return.

**Raises** `avendesora.PasswordError`

## 3.7.5 Utility Classes

These classes are used as account values, (see *Scripts*).

**class** `avendesora.Script` (`script='username: {username}, password: {passcode}'`)

Takes a string that contains attributes. Those attributes are expanded before being output. For example:

```
Script('username: {username}, password: {passcode}')
```

In this case, `{username}` and `{passcode}` are replaced by with the value of the corresponding account attribute. In addition to the account attributes, `{tab}` and `{return}` are replaced by a tab or carriage return character.

**Parameters** **script** (`str`) – The script.

## 3.8 Configuring

Avendesora is configured by way of a collection of files contained in the config directory (`~/config/avendesora`). This directory may contain the following files:

#### **accounts\_files:**

This file contains the list of known account files. The first file in the list is the default account file (this is where new accounts go by default). You can use the *new command* to add additional files to this list, but to delete account file you must manually edit this file and remove them from the list.

**config** and **config.doc**:

You control the behavior of Avendesora through a collection of settings that are specified in *config*. The available settings and their default values are documented in *config.doc*. Generally you only place values in *config* if you would like to change them from their default value. In that way, you will get the latest values for all other settings when you update Avendesora.

**hashes**:

One of the risks in using a password generator is that changed in the code can result in the passwords changing. Thus there is a risk that when you upgrade Avendesora that your passwords will change. Avendesora provides the *archive* and *changed* commands to help detect these situations. It also keeps hashes of several key parts of the code that if changed could result in the passwords changing. When Avendesora runs, it recomputes these hashes on itself and compares them to the hashes saved in this file. If any of the hashes have changed a warning message is produced, which can alert you to changes that you might have otherwise missed.

It is normal that these hashes change when the program is updated. When you see the message that the hashes have changed you should run the *changed command* to assure that none of your generated secrets have changed. This assumes that you have created an archive file and kept it up to date.

**stealth\_accounts**:

This file contains the definitions of the available stealth accounts. Stealth accounts allow you to create passwords for accounts that are not kept in an account file.

*account files*:

A file containing a collection of related accounts. All accounts in a file share a common master seed.

*archive file*:

This file contains all known accounts with any generated secrets expanded. It is used to identify account values that may have inadvertently changed.

*log file*:

The log file is created after each invocation of Avendesora. It provides details about the run that can help understand what happened during the run, which can help you resolve issues when things go wrong. This file can leak account information, and so it is best if it is encrypted.

### 3.8.1 Settings

The settings are documented in *config.doc*, and can be overwritten by specifying the desired values in the *config* file (found in `~/config/avendesora`). The available settings are:

**log\_file** = log.gpg:

The desired location of the log file (relative to config directory). Adding a suffix of .gpg or .asc causes the file to be encrypted (otherwise it can leak account names). Use None to disable logging.

**archive\_file** = archive.gpg:

The desired location of the archive file (relative to config director). End the path in .gpg or .asc. Use None to disable archiving.

**previous\_archive\_file** = archive.prev.gpg:

The existing archive file is renamed to this name when updating the archive file. This could be helpful if the archive file is somehow corrupted.

**archive\_stale** = 1:

The archive file is considered stale if it is this many days older than the most recently updated account file.

**default\_field** = 'passcode password passphrase':

The name of the field to use for the *value command* when one is not given. May be a space separated list of names, in which case the first that is found is used.

**default\_vector\_field** = 'questions':

The name of the field to use when an integer is given as the argument to the *value command*. In this case the field is expected to be a list and the argument is taken to be the index of the desired value. For example, if `default_vector_field` is 'question' and the argument given with the *value command* is 1, then `question[1]` is produced.

**credential\_ids** = 'username email':

A string that contains the field names (space separated) that should be considered by the *credentials command* for the account identity.

**credential\_secrets** = 'passcode password passphrase':

A string that contains the field names (space separated) that should be considered by the *credentials command* for the primary account secret.

**display\_time** = 60:

The number of seconds that the secret will be displayed before it is erased when writing to the TTY or clipboard.

**encoding** = 'utf-8': The unicode encoding to use when reading or writing files.

**edit\_account:**

The command used when editing an account. The command is given as a list of strings. The strings may contain {filepath} and {account}, which are replaced by the path to the file and the name of the account.

**edit\_template:**

The command used when creating a new account that has been initialized with a template. The command is given as a list of strings. The strings may contain {filepath}, which is replaced by the path to the file.

**browsers:**

A dictionary containing the supported browsers. For each entry the key is the name to be used for the browser, and the value is a string that contains the command that invokes the browser. The value may contain {url}, which is replaced by the URL to open.

**default\_browser:**

The name of the default browser. This name should be one of the keys in the browsers dictionary.

**default\_protocol** = 'https':

The default protocol to use for a URL if the protocol is not specified in the requested URL. Generally this should be 'https' or 'http', though 'https' is recommended.

**config\_dir\_mask** = 0o077:

An integer that determines if the permissions of *Avendesora* configuration directory (`~/config/avendesora`) are too loose. If they are, a warning is printed. A bitwise *and* operation is performed between this value and the actual file permissions, and if the result is nonzero, a warning is printed. Set to 0o000 to disable the warning. Set to 0o077 to generate a warning if the configuration directory is readable or writable by the group or others. Set to 0o007 to generate a warning if the directory is readable or writable by others.

**label\_color** = 'blue':



The color of the label use by the value and values commands. Choose from black, red, green, yellow, blue, magenta, cyan, white.

**highlight\_color** = 'magenta':

The color of the highlight use by the value and values commands. Choose from black, red, green, yellow, blue, magenta, cyan, white.

**color\_scheme** = 'dark':

The color scheme used for the label color. Choose from dark, light. If the shell background color is light, use dark.

**use\_pager = True:** Use a external program to break long output into pages. May be either a boolean or a string. If a string the string is taken to be a command line use to invoke a paging program (like 'more'). If True, the program name is taken from the PAGER environment variable if set, or 'less' is used if not set. If False, a paging program is not used.

**selection\_utility = 'gtk':** Which utility should be used when it becomes necessary for you to interactively make a choice. Two utilities are available: *gtk*, the default, and *dmenu*.

*gtk* is the built-in selection. When needed it pops a small dialog box in the middle of the screen. You can use the 'j' and 'k' to navigate to your selection and 'l' to make the selection or 'h' to cancel. Alternately you can use the arrow keys and Enter and Esc to navigate, select, and cancel.

*dmenu* is an external utility, and must be installed. With *dmenu* you type the first few letters of your selection to highlight it, then type 'Enter' to select or 'Esc' to cancel.

**verbose** = False:

Set this to True to generate additional information in the log file that can help debug account discovery issues. Normally it should be False to avoid leaking account information into log file. This is most useful when debugging account discovery, and in that case this setting has largely been superseded by the use of the `-title` and `-verbose` command line options.

**account\_templates:**

The available account templates. These are used when creating new accounts. The templates are given as a dictionary where the key is the name of the template and the value is the template itself. The template is passed through `textwrap.dedent()` to remove any leading white space. Any lines that begin with '#' Avendesora: ' represent comments that can contain instructions to the user. They will are removed when the account is created.

**default\_account\_template** = 'bank'

The default account template that is used when creating a new account and the user does not specify a template name.

**gpg\_ids:**

The GPG ID or IDs to use by default when creating encrypted files (the archive and account files).

**gpg\_armor = 'extension':** In the GPG world, armoring a file means converting it to simple ASCII. Choose between 'always', 'never' and 'extension' (.asc: armor, .gpg: no).

**gpg\_home** = ~/.gnupg:

This is your GPG home directory. By default it will be ~/.gnupg.

**gpg\_executable** = /usr/bin/gpg2:

Path to the *gpg2* executable.

**xdotool\_executable** = /usr/bin/xdotool:

Path to the *xdotool* executable.

**xsel\_executable** = /usr/bin/xsel:

Recommend '/usr/bin/xsel -p' if you wish to use mouse middle click. Recommend '/usr/bin/xsel -b' if you wish to use mouse right click then paste.

**dmenu\_executable** = /usr/bin/dmenu:

Path to the *dmenu* executable. *Avendesora* can be configured to use *dmenu* as selection utility rather than built-in *gtk* version.

## 3.9 Python API

### 3.9.1 A Simple Example

You can access account information from *Avendesora* using Python using a simple relatively high-level interface as shown in this example:

```
from avendesora import PasswordGenerator, PasswordError
from inform import display, fatal, os_error
from shlib import Run
from pathlib import Path

try:
    pw = PasswordGenerator()
    account = pw.get_account('mybank')
    name = account.get_value('name')
    username = account.get_username()
    passcode = account.get_passcode()
    url = account.get_value('ofxurl')
except PasswordError as e:
    e.terminate()

try:
    curl = Run(
        f'curl -K - {url!s}',
        stdin = f'user="{username!s}:{passcode!s}"',
        modes='sOEW0'
    )
    Path(f'{name!s}.ofx').write_text(curl.stdout)
except OSError as e:
    fatal(os_error(e))
```

Basically, the approach is to open the password generator, open an account, and then access values of that account. The various components of the *Avendesora* programming interface are described next.

### 3.9.2 Components

This section documents the programming interface for *Avendesora*. You can view the *Avendesora* source code, particularly `avendesora.command`, for further examples on how to use this interface.

## PasswordGenerator Class

This is the entry class to *Avendesora*. It is the only class you need instantiate directly. By instantiating it you cause *Avendesora* to read the user's account files.

**class** `avendesora.PasswordGenerator` (*init=False, gpg\_ids=None, warnings=True*)

Initializes the password generator. You should pass no arguments unless you are creating the user's Avendesora data directory.

Calling this class causes Avendesora to open all the various account files and returns an object that allows you access to the accounts. Specifically you can use the `get_account()` or `all_accounts()` methods to access an account or all the accounts.

### Parameters

- **init** (*bool*) – Create user's directory.
- **gpg\_ids** (*list of strings*) – List of GPG identities to use when creating user's directory.

**Raises** `avendesora.PasswordError` – Indicates an issue opening the user's accounts.

**all\_accounts** ()

Iterate through all accounts.

**challenge\_response** (*name, challenge*)

Generate a response to a challenge.

Given the name of a master seed (actually the basename of the file that contains the master seed), returns an identifying response to a challenge. If no challenge is provided, one is generated based on the time and date. Returns both the challenge and the expected response as a tuple.

### Parameters

- **name** (*str*) – The name of the master seed.
- **challenge** (*str*) – The challenge (may be empty).

**discover\_account** (*url=None, title=None, verbose=False*)

Discover the account from the environment.

Examine the environment and return the script that matches (the script is initialized, and so contains a pointer to the right account). If more than one account/secret matches, user is queried to resolve the ambiguity.

### Parameters

- **url** (*str*) – Specifying the URL short-circuits the processing of the title that is used to find the URL.
- **title** (*str*) – Override the window title. This is used for debugging.
- **verbose** (*bool*) – Run the discovery process in verbose mode (adds more information to log file that can help debug account discovery).

**Raises** `avendesora.PasswordError` – There is no account that matches the given environment.

**find\_accounts** (*target*)

Find accounts with names or aliases that contain a substring.

**Parameters** **target** (*str*) – The desired substring.

**Returns** Iterates through matching accounts.

**Return type** *avendesora.Account*

**get\_account** (*name*, *request\_seed=False*, *stealth\_name=None*)

Return a specific account.

**Parameters**

- **name** (*str*) – Looks up an account by name and returns it. This name must match an account name or an account alias. The matching algorithm ignores case and treats dash and underscore as equivalent.
- **request\_seed** (*str or bool*) – If specified an additional seed is provided to the account (see: *misdirection*). It may be specified as a string, in which case it is used as the seed. Otherwise if true, the seed it requested directly from the user.
- **stealth\_name** (*str*) – The name used as the account name if the account is a stealth account.

**Returns** An account. The class itself is returned, and not an instance of the class.

**Return type** *avendesora.Account*

**Raises** *avendesora.PasswordError* – There is no account that matches the given name.

**search\_accounts** (*target*)

Find accounts with values that contain a substring.

**Parameters** **target** (*str*) – The desired substring.

**Returns** Iterates through matching accounts.

**Return type** *avendesora.Account*

## Account Class

**class** *avendesora.Account*

Class that holds all the information specific to an account.

Add desired account information as attributes of the class.

**classmethod** **get\_composite** (*name*)

Get field value given a field name.

A lower level interface than *get\_value()* that given a name returns the value of the associated field, which may be a scalar (string or integer) or a composite (array of dictionary). Unlike *get\_value()*, the actual value is returned, not a object that contains multiple facets of the value.

**Parameters** **name** (*str*) – The name of the field.

**Returns** The requested value.

**classmethod** **get\_fields** (*all=False*)

Iterate through fields.

Iterates through the field names.

Example:

```
for name, keys in account.get_fields():
    if keys:
        display(name + ':')
        for key, value in account.get_values(name):
            display(indent(
                value.render({'{k} {d}: {v}', '{k}: {v}'))
```

```

        ))
    else:
        value = account.get_value(name)
        display(value.render('{n}: {v}'))

```

**Parameters** `all` (*bool*) – If False, ignore the tool fields.

**Returns** A pair (2-tuple) that contains both field name and the key names. None is returned for the key names if the field holds a scalar value.

**classmethod** `get_name()`

Get account name.

**Returns** Returns the primary account name. This is generally the class name converted to lower case unless it was overridden with the NAME attribute.

**classmethod** `get_passcode()`

Get the passcode.

Like `get_value()`, but tries the `credential_secrets` in order and returns the first found. `credential_secrets` is an Avendesora configuration setting that by default is `password`, `passphrase`, and `passcode`.

**Returns** The passcode.

**classmethod** `get_scalar(name, key=None, default=False)`

Get field Value given a field name and key.

A lower level interface than `get_value()` that given a name and perhaps a key returns a scalar value. Also takes an optional default value that is returned if the value is not found. Unlike `get_value()`, the actual value is returned, not a object that contains multiple facets of the value.

The `name` is the field name, and the `key` would identify which value is desired if the field is a composite. If default is False, an error is raised if the value is not present, otherwise the default value itself is returned.

**Parameters**

- **name** (*str*) – The name of the field.
- **key** (*str or int*) – The key for the desired value (should be None for scalar values).
- **default** – The value to return if the requested value is not available.

**Returns** The requested value.

**classmethod** `get_username()`

Get the username.

Like `get_value()`, but tries the `credential_ids` in order and returns the first found. `credential_ids` is an Avendesora configuration setting that by default is `username` and `email`.

**Returns** The username or email address.

**classmethod** `get_value(field=None)`

Get account value.

Return value from the account given a user friendly identifier or script. User friendly identifiers include:

*None*: value of default field

*name*: scalar value

*name.key* or *name[key]*:

member of a dictionary or array

key is string for dictionary, integer for array

Scripts are simply strings with embedded attributes. Ex: `'username: {username}, password: {passcode}'`

**Parameters** `field` (*str*) – Field identifier or script.

**Returns** the desired value.

**Return type** `avendesora.AccountValue`

**classmethod** `get_values` (*name*)

Iterate through the values for a field.

**Parameters** `name` (*str*) – The name of the field.

**Returns** Returns a pair (2-tuple) that contains the key and the value given as an `avendesora.AccountValue` for each of the values. If the value is a scalar, the key is `None`.

## AccountValue Class

## PasswordError Exception

**exception** `avendesora.PasswordError` (*\*args, \*\*kwargs*)

Password error.

This exception subclasses `Inform.Error`.

This exception subclasses `inform.Error`.

**get\_culprit** ()

Get exception culprit.

If the `culprit` keyword argument was specified as a string, it is returned. If it was specified as a collection, the members are converted to strings and joined with `culprit_sep`. The resulting string is returned.

**get\_message** (*template=None*)

Get exception message.

**Parameters** `template` (*str*) – This argument is treated as a format string and is passed both the unnamed and named arguments. The resulting string is treated as the message and returned.

If not specified, the `template` keyword argument passed to the exception is used. If there was no `template` argument, then the positional arguments of the exception are joined using `sep` and that is returned.

**Returned:** The formatted message without the culprits.

**render** (*template=None*)

Convert exception to a string for use in an error message.

**Parameters** `template` (*str*) – This argument is treated as a format string and is passed both the unnamed and named arguments. The resulting string is treated as the message and returned.

If not specified, the `template` keyword argument passed to the exception is used. If there was no `template` argument, then the positional arguments of the exception are joined using `sep` and that is returned.

**report** (*template=None*)

Report exception.

The `inform.error()` function is called with the exception arguments.

**Parameters** `template` (*str*) – This argument is treated as a format string and is passed both the unnamed and named arguments. The resulting string is treated as the message and returned.

If not specified, the `template` keyword argument passed to the exception is used. If there was no `template` argument, then the positional arguments of the exception are joined using `sep` and that is returned.

**terminate** (`template=None`)

Report exception and terminate.

The `inform.fatal()` function is called with the exception arguments.

**Parameters** `template` (*str*) – This argument is treated as a format string and is passed both the unnamed and named arguments. The resulting string is treated as the message and returned.

If not specified, the `template` keyword argument passed to the exception is used. If there was no `template` argument, then the positional arguments of the exception are joined using `sep` and that is returned.

**with\_traceback** ()

`Exception.with_traceback(tb)` – set `self.__traceback__` to `tb` and return `self`.

### 3.9.3 Example: Displaying Account Values

The following example prints out all account values for account whose name are found in a list.

```
from avendesora import PasswordGenerator
from inform import display, indent, Error

accounts = ['bank', 'credit-union', 'brokerage']

try:
    pw = PasswordGenerator()

    for account_name in accounts:
        account = pw.get_account(account_name)
        description = account.get_scalar('desc', None, account_name)
        display(description, len(description)*' ', sep='\n')

        for name, keys in account.get_fields():
            if keys:
                display(name + ':')
                for key, value in account.get_values(name):
                    display(indent(
                        value.render('{k} {d}: {v}', '{k}: {v}'))
                    ))
            else:
                value = account.get_value(name)
                display(value.render('{n}: {v}'))
        display()
except Error as e:
    e.terminate()
```

### 3.9.4 Example: Add SSH Keys

This example adds SSH keys to your SSH agent. It uses *pexpect* to manage the interaction between this script and *ssh-add*.

```
#!/usr/bin/env python3
"""
Add SSH keys

Add SSH keys to SSH agent.
The following keys are added: {keys}.

Usage:
    addsshkeys [options]

Options:
    -v, --verbose    list the keys as they are being added
"""
# This assumes that the Avendesora account that contains the ssh key's
# passphrase has a name or alias of the form <name>-ssh-key. It also assumes
# that the account contains a field named 'keyfile' or 'keyfiles' that contains
# an absolute path or paths to the ssh key files in a string.

from avendesora import PasswordGenerator, PasswordError
from inform import Inform, codicil, error, fatal, narrate
from docopt import docopt
from pathlib import Path
import pexpect

SSHkeys = ['primary', 'github', 'backups']
SSHadd = 'ssh-add'

cmdline = docopt(__doc__.format(keys = ', '.join(SSHkeys)))
Inform(narrate=cmdline['--verbose'])

try:
    pw = PasswordGenerator()
except PasswordError as e:
    e.terminate()

for key in SSHkeys:
    name = key + '-ssh-key'
    try:
        account = pw.get_account(name)
        passphrase = account.get_passcode().value
        if account.has_field('keyfiles'):
            keyfiles = account.get_value('keyfiles').value
        else:
            keyfiles = account.get_value('keyfile').value
        for keyfile in keyfiles.split():
            path = Path(keyfile).expanduser()
            narrate('adding.', culprit=keyfile)
            try:
                sshadd = pexpect.spawn(SSHadd, [str(path)])
                sshadd.expect('Enter passphrase for %s: ' % (path), timeout=4)
                sshadd.sendline(passphrase)
                sshadd.expect(pexpect.EOF)
                sshadd.close()
            except:
                pass
```



```

        response = sshadd.before.decode('utf-8')
        if 'identity added' in response.lower():
            continue
    except (pexpect.EOF, pexpect.TIMEOUT):
        pass
    error('failed.', culprit=path)
    codicil('response:', sshadd.before.decode('utf8'), culprit=SSHadd)
    codicil('exit status:', sshadd.exitstatus, culprit=SSHadd)
except PasswordError as e:
    fatal(e, culprit=path)

```

### 3.9.5 Example: Postmortem Letter

This is a program that generates messages for a person's children and business partners. It is assumed that these messages would be placed into a safe place to be found and read upon the person's death.

It examines all accounts looking for a special field, *postmortem\_recipients*. If the field exists, then that account is included in the file of accounts sent to that recipient. The script also looks for another special field, *estimated\_value*. It includes this value in the message and prints the values to the standard output when generating the messages. This gives you a chance to review the values and update them if they are stale. The generated files are encrypted so that only the intended recipients can read them.

```

#!/usr/bin/env python3

from avendesora import PasswordGenerator, PasswordError
from inform import Error, display, indent, os_error, terminate, warn
import gnupg

me = 'morgase@andor.gov'
recipients = dict(
    kids='galad@trakand.name gawyn@trakand.name elayne@trakand.name',
    partners='taringail.damodred@andor.gov',
)

try:
    pw = PasswordGenerator()
    accounts = {}

    # scan accounts and gather information for recipients
    for account in pw.all_accounts():

        # summarize account values
        value = account.get_scalar('estimated_value', default=None)
        postmortem_recipients = account.get_scalar('postmortem_recipients',
        ↪default=None)
        if value and postmortem_recipients:
            display(f'{account.get_name()}: {value}')
        elif value:
            warn(f'{account.get_name()}: no recipients.')
        elif not postmortem_recipients:
            continue

    # gather information for recipients
    for recipient in recipients:
        if recipient in postmortem_recipients:
            account_name = account.get_name()

```

```
description = account.get_scalar('desc', None, account_name)
lines = [description, len(description)*'=']

for name, keys in account.get_fields():
    if name == 'postmortem_recipients':
        continue
    if keys:
        lines.append(name + ':')
        for key, value in account.get_values(name):
            lines += indent(
                value.render({'k}: {v}', '{k}: {v}'))
            ).split('\n')
    else:
        value = account.get_value(name)
        lines += value.render('{n}: {v}').split('\n')
if recipient not in accounts:
    recipient_accounts = []
    accounts[recipient] = recipient_accounts
recipient_accounts.append('\n'.join(lines))

# generate encrypted files than contain about accounts for each recipient
gpg = gnupg.GPG(gpgbinary='gpg2')
for recipient, idents in recipients.items():
    if recipient in accounts:
        content = accounts[recipient]
        num_accounts = len(content)
        encrypted = gpg.encrypt(
            '\n\n\n'.join(content),
            idents.split() + me.split()
        )
        if not encrypted.ok:
            raise Error(
                'unable to encrypt:', encrypted.stderr, culprit=recipient
            )
        try:
            filename = recipient + '.gpg'
            with open(filename, 'w') as file:
                file.write(str(encrypted))
            display(f'contains {num_accounts} accounts.', culprit=filename)
        except OSError as e:
            raise Error(os_error(e))
    else:
        warn('no accounts found.', culprit=recipient)

except KeyboardInterrupt:
    terminate('Killed by user.')
except (PasswordError, Error) as e:
    e.terminate()
```

### 3.9.6 Example: Net Worth

If you have added *estimated\_value* to all of your accounts that hold significant value as proposed in the previous example, then the following script will summarize the values and estimate your net worth:

```
#!/usr/bin/env python3
"""Networth

Usage:
    networth [options]

Options:
    -v, --verbose output actual estimated value text rather than just the value
"""

from avendesora import PasswordGenerator, PasswordError
from inform import display, terminate
from quantiphy import Quantity
from docopt import docopt
Quantity.set_prefs(prec=2)

cmdline = docopt(__doc__)
verbose = cmdline['--verbose']

try:
    pw = PasswordGenerator()

    # scan accounts and gather values for appropriate accounts
    amounts = {}
    width = 0
    total = 0
    for account in pw.all_accounts():
        text = account.get_scalar('estimated_value', default=None)
        if text:
            components = text.split()
            value = Quantity(components[0])
            if verbose:
                to_output = text
            else:
                to_output = value
            if value.units in ['$ ', 'USD']:
                total += value
            else:
                to_output = f'{to_output} (not in total)'
            account_name = account.get_name()
            width = max(width, len(account_name))
            amounts[account_name] = to_output
    total = Quantity(total, "$")

    # display the account values
    for key, val in amounts.items():
        display(f'{key:>{width+2}s}: {val}')
    display(f'{"TOTAL":>{width+2}s}: {total}')

except KeyboardInterrupt:
    terminate('Killed by user.')
except PasswordError as e:
    e.terminate()
```

This script assumes that the account value with units are the first thing in the *estimated\_value* string. It is common to combine the estimated value with the date it was last updated. Something like this:

```
estimated_value = '$11k in January 2018'
```

The text before the first whitespace is considered the value, and if it has units of dollars it will be added to the reported total. Here is a typical output:

```
mint: $19k
betterment: $22k
chase: $12k
southwest: 78kmiles (not in total)
TOTAL: $53k
```

## 3.10 Examples

### 3.10.1 Challenge Questions

Websites originally used challenge questions to allow you to re-establish your identity if you lose your user name or password, so it was enough to simply save the answers so that they were available if needed. But now many websites require you to answer the challenge questions if the site does not recognize your browser because your cookie expired or was deleted. As such, people need to answer their challenge questions with much more frequency. Generally the site will save your answers to 4 or 5 challenge questions, and will present you with 1 or 2 at random. You must answer them correctly before you are allowed to login. To accommodate these needs, *Avendesora* saves the challenge questions and either stores or generates the answers. It also makes it easy for you to autotype the answer to any of your questions.

The following shows how to configure an account to support challenge questions.

```
class BankOfAmerica(Account):
    aliases = 'boa bankamerica'
    username = 'matrim'
    passcode = PasswordRecipe('12 2l 2u 2d 2s')
    questions = [
        Question('elementary school?'),
        Question('favorite foreign city?'),
        Question('first pet?'),
        Question('what year was your father born?'),
        Question('favorite movie?'),
    ]
    discovery = [
        RecognizeURL(
            'https://www.bankofamerica.com/',
            script='{username}{tab}{passcode}{return}'
        ),
        RecognizeURL(
            'https://secure.bankofamerica.com',
            script='{questions}{tab}',
        ),
    ]
```

In this case 5 questions are supported. When you are first required to set up your challenge questions the website generally presents you with 20 or 30 to choose from. Simply choose the first few and add them to your account.

Then use the *value command* to generate the answers and copy them into the website. You need not enter the questions into *Avendesora* exactly, but once you provide your website with the generated answers you must not change the questions in any way because doing so would change the answers. Finally, the first time you are required to enter answers to the challenge questions, take note of the URL and add a discovery entry that matches the url and generates the

questions. In most cases you will not be able to specify a single question, so simply specify the array and *Avendesora* will allow you to choose a particular question when you request an answer. Specifically, when the website takes you to the challenge question page, click in the field for the first answer and type the hotkey that runs *Avendesora* in autotype mode. *Avendesora* should recognize the page and allow you to identify the question. It will then autotype the answer into the field and then move to the next field. Alternately, if you terminate the script with '{return}' rather than '{tab}', it will take you to the next page.

In some cases the website makes you choose from a fixed set of answers. In this case you would save the answer with the question as follows:

```
class BankOfAmerica(Account):
    ...
    questions = [
        Question('elementary school?', answer='MLK Elementary'),
        Question('favorite foreign city?', answer='Kashmir'),
        Question('first pet?', answer='Spot'),
        Question('what year was your father born?', answer='1950'),
        Question('favorite movie?', answer='A boy and his dog'),
    ]
    ...
```

### 3.10.2 Two Page Authentication

A new trend in recent years is websites that use two-page authentication. This is where you enter your user name or email on one page, you then submit it and get another page that you use to enter your password, which requires a second submission. Google moved to two-page authentication some time ago, and now Amazon seems to be switching as well. Originally this was intended as an anti-phishing strategy. After entering your user name you are shown a site image and phrase that you can use to confirm that you are logging in to the correct site. This is unnecessary when using *RecognizeURL* because it will only enter your user name and password if the URL is correct. Recently however, sites have dispensed with the site image and phrase, but still spread the login in process over two pages. It is not clear why they do this. There does not seem to be any security benefit. In fact it acts to reduce security by making it more difficult to use a password manager. Unfortunately this is all too common. Companies talk a good game when it comes to security, but all too often employ practices that are antithetical to good security.

There are two approaches to handling two-page authentication in *Avendesora*. The first would be to split the account discovery into two steps. For example:

```
class Gmail(Account):
    aliases = 'email'
    username = 'matrim.cauthon'
    passcode = Passphrase()
    urls = 'https://accounts.google.com/signin/v2/identifier'
    discovery = [
        RecognizeURL(
            'https://accounts.google.com/ServiceLogin/identifier',
            'https://accounts.google.com/signin/v2/identifier',
            script='{username}{return}',
            name='username',
        ),
        RecognizeURL(
            'https://accounts.google.com/signin/v2/sl/pwd',
            script='{passphrase}{return}',
            name='passcode',
        ),
    ]
```

Notice that there are two instances of *RecognizeURL*, both looking for different URLs. You would trigger *Avendesora* to enter the user name, then trigger it again to enter the passcode. This is the best case situation in that the URLs for each page are distinct. However, some sites make it difficult to distinguish what is being asked for just by looking at the URL. Amazon is one of those:

```
class Amazon(Account):
    email = 'matrim@tworivers.com'
    passcode = Passphrase()
    discovery = [
        RecognizeURL(
            'https://www.amazon.com/ap/signin',
            script = '{email}{return}',
            name = 'email',
        ),
        RecognizeURL(
            'https://www.amazon.com/ap/signin',
            script = '{passcode}{return}',
            name = 'passcode',
        ),
    ]
```

Notice that the URL is the same for both recognizers, which causes *Avendesora* to ask you which you want each time you request either. A variation on this is to have different URLs for each page, but one URL is a subset of the other. For example, 'andorsavings.com/signin' and 'andorsavings.com/signin/pwd'. By default *Avendesora* will offer both when it comes time to enter the password, but adding 'exact\_path=True' to the username recognizer causes *Avendesora* to be more selective.

The second approach is use just one recognizer that outputs both the user name and password, but to add a delay between them. For example:

```
class Amazon(Account):
    email = 'amazon@shalmirane.com'
    passcode = Passphrase()
    discovery = [
        RecognizeURL(
            'https://www.amazon.com/ap/signin',
            script = '{email}{return}{sleep 2}{passcode}{return}',
            name = 'both',
        ),
    ]
```

In this way you would only need to trigger *Avendesora* once. You might have to adjust the sleep time to be able to log in reliably.

### Chrome

The Chrome browser seems to have a bug that can interfere with its use with account discovery. In the two step process used when logging in, the site might pre-fill-in your user name so you do not have to enter it explicitly, you just have to click *next*. Chrome then takes you to the page where you are expected to enter your password, however when it does so it does not update the window title to match the new page. Then *Avendesora* sees the wrong URL and either enters the wrong thing or does not recognize the page. To work around this bug, you must refresh the page when you land on the password page before activating *Avendesora*.

### 3.10.3 Wireless Router

Wireless routers typically have two or more secrets consisting of the admin password and the passwords for one or more wireless networks. For example, the router in this example supports two networks, a privileged network that allows connections to the various devices on the local network and the guest network that only access to the internet. In this case all three employ pass phrases. The admin password is held in *passcode* and the network names and passwords are held in the *network\_passwords* array. To make the information about each network easy to access from the command line, two scripts are defined, *guest* and *privileged*, and each produces both the network name and the network password for the corresponding networks.

Secret discovery handles two distinct cases. The first case is when from within your browser you navigate to your router (ip=192.168.1.1). In this situation, the URL is matched and the script is run that produces the administrative username and password. The second case is when you attempt to connect to a wireless network and a dialog box pops up requesting the SSID and password of the network you wish to connect to. Running *xwininfo* shows that the title of the dialog box is 'Wi-Fi Network Authentication Required'. When this title is seen, both the title recognizers match, meaning that both the privileged and the guest credentials are offered as choices.

```
class NetgearAC1200_WirelessRouter(Account):
    NAME = 'home-router'
    aliases = 'wifi'
    admin_username = 'admin'
    admin_password = Passphrase()
    default = 'admin_password'
    networks = ["Occam's Router", "Occam's Router (guest)"]
    network_passwords = [Passphrase(), Passphrase()]
    privileged = Script('SSID: {networks.0}{return}password: {network_passwords.0}')
    guest = Script('SSID: {networks.1}{return}password: {network_passwords.1}')
    discovery = [
        RecognizeURL(
            'http://192.168.1.1',
            script='{admin_username}{tab}{admin_password}{return}'
        ),
        RecognizeTitle(
            'Wi-Fi Network Authentication Required',
            script='{networks.0}{tab}{network_passwords.0}{return}',
            name='privileged network'
        ),
        RecognizeTitle(
            'Wi-Fi Network Authentication Required',
            script='{networks.1}{tab}{network_passwords.1}{return}',
            name='guest network'
        ),
    ]
    model_name = "Netgear AC1200 wireless router"
```

### 3.10.4 Credit Card Information

Many websites offer to store your credit card information. Of course, we have all heard of the massive breaches that have occurred on such websites, often resulting in the release of credit card information. So all careful denizens of the web are reluctant to let the websites keep their information. This results in you being forced into the tedious task of re-entering this information.

*Avendesora* can help with this. If you have a website that you find yourself entering credit card information into routinely, then you can use the account discovery and autotype features of *Avendesora* to enter the information for you.

For example, imagine that you have a Citibank credit card that you use routinely on the Costco website. You can configure *Avendesora* to automatically enter your credit card information into the Costco site with by adding an account discovery entry to your Citibank account as follows:

```
class CostcoCitiVisa(Account):
    aliases = 'citi costcovisa'
    username = 'mcauthon'
    email = 'matrim@gmail.com'
    account = '1234567889012345'
    expiration = '03/2019'
    cvv = '233'
    passcode = PasswordRecipe('12 2u 2d 2s')
    verbal = Question('Favorite pet?', length=1)
    questions = [
        Question("Fathers profession?"),
        Question("Last name of high school best friend?"),
        Question("Name of first pet?"),
    ]
    urls = 'https://online.citi.com'
    discovery = [
        RecognizeURL(
            'https://online.citi.com',
            script='{username}{tab}{passcode}{return}',
            name='login'
        ),
        RecognizeURL(
            'https://www.costco.com/CheckoutPaymentView',
            script='{account}{tab}{expiration}{tab}{cvv}{tab}Matrim Cauthon{return}',
            name='card holder information'
        ),
    ]
```

This represents a relatively standard *Avendesora* description of an account. Notice that it contains the credit card number (*account*), the expiration date (*expiration*) and the CVV number (*cvv*). This is raw information the autotype script will pull from. The credit card and the CVV values are sensitive information and should probably be concealed.

Also notice the two *avendesora.RecognizeURL* entries in *discovery*. The first recognizes the CitiBank website. The second recognizes the Costco check-out page. When it does, it runs the following script:

```
{account}{tab}{expiration}{tab}{cvv}{tab}Matrim Cauthon{return}
```

That script enters the account number, tabs to the next field, enters the expiration date, tabs to the next field, enters the CVV, tabs to the next field, enters the account holders name, and finally types return to submit the information (you might want to delete the {return} so that you have a chance to review all the information before you submit manually. Or you could continue the script and give more information, such as billing address.

Conceptually this script should work, but Costco, like many websites, uses Javascript helpers to interpret the fields. These helpers are intended to give you immediate feedback if you typed something incorrectly, but they are slow and can get confused if you type too fast. As is, the first one or two fields would be entered properly, but the rest would be empty because they were entered by *Avendesora* before the page was ready for them. To address this issue, you can put delays in the script:

```
{account}{tab}{sleep 0.5}{expiration}{tab}{sleep 0.5}{cvv}{tab}{sleep 0.5}Matrim_
↵Cauthon{return},
```

Now the account can be given in its final form. This differs from the one above in that the *account* and *cvv* values are concealed and the delays were added to the Costco script.



```

class CostcoCitiVisa(Account):
    aliases = 'citi costcovisa'
    username = 'mcauthon'
    email = 'matrim@gmail.com'
    account = Hidden('MTIzNCA1Njc4IDg5MDEgMjM0NQ==')
    expiration = '03/2019'
    cvv = Hidden('MjMz')
    passcode = PasswordRecipe('12 2u 2d 2s')
    verbal = Question('Favorite pet?', length=1)
    questions = [
        Question("Fathers profession?"),
        Question("Last name of high school best friend?"),
        Question("Name of first pet?"),
    ]
    discovery = [
        RecognizeURL(
            'https://online.citi.com',
            script='{username}{tab}{passcode}{return}',
            name='login'
        ),
        RecognizeURL(
            'https://www.costco.com/CheckoutPaymentView',
            script='{account}{tab}{sleep 0.5}{expiration}{tab}{sleep 0.5}{cvv}{tab}
↵{sleep 0.5}Matrim Cauthon{return}',
            name='card holder information'
        ),
    ]

```

This approach requires that you anticipate those sites into which you will enter the credit card information. Alternatively, you add a script to your credit card account that outputs the credit card information, and then run *Avendesora* in such a way that the credit card information into the webpage. To do this requires two things. First, add a script to the account that combines and outputs the credit card information. For example:

```
ccn = Script('{account}{tab}{cvv}{tab}')
```

In this case the amount of information requested is limited to increase the chance that the result will be compatible with a large number of websites. Then run *Avendesora* from the window manager:

```
Alt-F2 avendesora citi ccn
```

Here, Alt-F2 is the hot key Gnome uses to execute a command. This causes *Avendesora* to run the *ccn* script. Since *Avendesora* running from the window manager does not have access to a TTY it will instead mimic the keyboard and autotype the credit card information into the active window.

### 3.10.5 Swarm Accounts

You might find the need to have many accounts at one website, and for simplicity would like to share most of the account information. For example, you would share the URL and perhaps the password, but not the usernames.

You might wish to have multiple email addresses from a single email provider like gmail, or perhaps you would multiple accounts at a review site, like yelp.

In this case we give the list of account name in the *usernames* attribute. Then we use Python list comprehensions that use the *usernames* array to construct other values. That way to add a new account, you only need modify *usernames* and everything else is updated automatically.

```

class YandexMail(Account):
    aliases = 'yandex'
    usernames = 'rand.alThor aviendha rhuarc sorilea amys'.split()
    credentials = ' '.join(
        ['usernames.%d' % i for i in range(len(usernames))] + ['passcode']
    )
    email = [n + '@yandex.com' for n in usernames]
    passcode = PasswordRecipe('12 2u 2d 2s')
    questions = [
        Question('Surname of favorite musician?'),
    ]
    urls = 'https://mail.yandex.com'
    discovery = [
        RecognizeURL(
            'https://mail.yandex.com',
            script='{email[%s]}{tab}{passcode}{return}' % i,
            name=n,
        ) for i, n in enumerate(usernames)
    ]

```

Now, running the *credentials* command gives:

```

> avendesora yandex
usernames: rand.alThor
usernames: aviendha
usernames: rhuarc
usernames: sorilea
usernames: amys
passcode: B-F?i0z8GcDL

```

This example shows that the capabilities of the Python language can be used in the accounts files to increase the capabilities of *Avendesora* in unexpected ways.

### 3.10.6 Recognizing Shell Commands

Modern shells inform their terminal emulator of the currently running command. Modern terminal emulators then use that information to display the command in the window title. Generally this happens automatically, but if it is not working for you, you may have to manually configure your shell. Generally, you configure the shell by changing the value of the variable that sets the command prompt.

If your window manager is configured to not show window titles, you can still determine the title using *xwininfo*.

If your shell does not set the window title you can still use the window title to trigger *Avendesora* secrets recognition by explicitly setting the window title using *xdotool*. For example:

```

#!/bin/bash

original_title=`xdotool getactivewindow getwindowname`
xdotool getactivewindow set_window --name 'Home email'
mutt -F ~/.config/mutt/home
xdotool getactivewindow set_window --name "$original_title"

```

Once you have desired information in the window title, you can use the use *avendesora.RecognizeTitle* to trigger *Avendesora*. For example, you might use the following as the entry for you Linux password:

```

class Login(Account):
    desc = 'Linux login'

```

```
aliases = 'linux sudo'
passcode = Passphrase()
discovery = RecognizeTitle(
    'sudo *',
    script='{passcode}{return}'
)
```

You cannot use *Avendesora* to login to Linux, however once you have logged in you can use *Avendesora* to deliver your linux password to the sudo command.

An alternative to using window titles is to trigger *Avendesora* secrets recognition is to use *avendesora.RecognizeFile* as shown in *Account Discovery*.

### 3.11 Upgrading

Avendesora is primarily a password generator. As a result, there is always a chance that something could change in the password generation algorithm that causes the generated passwords to change. Of course, the program is thoroughly tested to assure this does not happen, but there is still a small chance that something slips through. To assure that you are not affected by this, you should archive your passwords before you upgrade with:

```
avendesora changed
avendesora archive
```

The *changed command* should always be run before an *archive command*. It allows you to review all the changes that have occurred so that you can verify that they were all intentional. Once you are comfortable, run the *archive command* to save all the changes. This creates a file (`~/config/avendesora/archive.gpg`) that contains all of your account information, including the secrets. Be sure to keep it safe.

Once you have created/updated your archive, you can upgrade Avendesora with:

```
pip install -upgrade --user avendesora
```

Finally, run:

```
avendesora version
```

to confirm that your version of *Avendesora* was updated and:

```
avendesora changed
```

to confirm that none of your generated passwords have changed.

It is a good idea to run ‘avendesora changed’ and ‘avendesora archive’ on a routine basis to keep your archive up to date. Doing so can help protect you against common mistake you might make.

Upon updating you may find that Avendesora produces a message that a ‘hash’ has changed. This is an indication that something has changed in the program that could affect the generated secrets. Again, care is taken when developing Avendesora to prevent this from happening. But it is an indication that you should take extra care. Specifically you should follow the above procedure to assure that the value of your generated secrets have not changed. Once you have confirmed that the upgrade has not affected your generated secrets, you should follow the directions given in the warning and update the appropriate hash contained in `~/config/avendesora/.hashes`.

## 3.12 Releases

### 1.0 (2017-01-01):

- Initial production release.

### 1.1 (2016-01-03):

### 1.2 (2017-01-05):

### 1.3 (2017-01-08):

- Warn about duplicate account names.

### 1.4 (2017-01-09):

- Improved error reporting on encrypted files.
- Added `RecognizeFile()`.

### 1.5 (2017-03-01):

- Fixed bug in account discovery for URLs.
- Added `get_composite`, renamed `get_field` to `get_scalar`.

### 1.6 (2017-04-07):

- Fix issues in sleep feature in autotype scripts.

### 1.7 (2017-06-01):

- add *credentials command*.

### 1.8 (2017-11-23):

- Created the manual.
- Use keyboard writer if there is no access to TTY.
- Shifted to skinny config file.
- Warn the user if the archive is missing or stale.
- Improved `get_value()`, added `add get_values()`, `add get_fields()`.
- Canonicalize names.
- Allow account name to be given even if TTY is not available.
- Allow `default_field` to be a list.
- Add render method to `AccountValue`.
- Changed the way multiple `gpg` ids are specified.
- Improved *browse command*.
- Added `shift_sort` to password generators.
- Added *log command*.
- Added *phonetic command*.
- Added browser version of *help command*.

It is recommended that in this release you trim your `~/config/avendesora/config` file to only include those settings that you explicitly wish to override.

### 1.9 (2017-12-25):

- Adds *one-time passwords* (an alternative to Google Authenticator).
- Added 'vc' command as an alias for 'value -clipboard'.

**1.10 (2018-02-18):**

- Added support for *qutebrowser*.

**Latest Development Version:**

Version: 1.10.5

Released: 2018-03-28

- Added *is\_secret* argument to Secret classes.
- Added support for *dmenu* as alternative to built-in selection utility.
- Add -delete option to log command.



## CHAPTER 4

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`





## Symbols

2FA, 32

### A

abbreviations, 16

abraxas, 38

Account (class in avendesora), 64

account discovery, 26

account files, 59

account\_templates setting, 61

accounts\_files file, 58

add command, 39, 46

adding account, 23

adding account file, 23

all\_accounts() (avendesora.PasswordGenerator method), 63

alphabet command, 43

alphabet, phonetic, 38

ALPHANUMERIC (avendesora attribute), 53

amazon, 73

archive command, 39

archive file, 39, 40, 59

archive\_file setting, 59

archive\_stale setting, 59

Authy, 32

### B

bank account, 20

BirthDate (class in avendesora), 51

browse, 31

add, 39

browse command, 39

browser configuration, 26

browsers, 27

browsers setting, 60

### C

challenge questions, 29, 72

challenge response, 37

challenge\_response() (avendesora.PasswordGenerator method), 63

changed

add, 40

changed command, 40

Chrome, 74

Chrome browser, 27

collaboration, 36

color\_scheme setting, 61

command

add, 39

alphabet, 43

archive, 39

conceal, 40

credentials, 41

edit, 41

find, 42

help, 42

identity, 42

initialize, 42

log, 43

login, 41

new, 43

phonetic, 43

reveal, 44

search, 44

value, 44

values, 45

conceal command, 40

config file, 59

config.doc file, 59

config\_dir\_mask setting, 60

configuring, 12

configuring browser, 26

configuring window manager, 13

confirming identity, 37

credential\_ids setting, 60

credential\_secrets setting, 60

credentials command, 41

credit cards, 75

## D

default\_account\_template setting, 61  
 default\_browser setting, 60  
 default\_field setting, 60  
 default\_protocol setting, 60  
 default\_vector\_field setting, 60  
 deleting account, 23  
 deleting account file, 23  
 DIGITS (avendesora attribute), 53  
 discover\_account() (avendesora.PasswordGenerator method), 63  
 discovery, 26  
 display\_time setting, 60  
 DISTINGUISHABLE (avendesora attribute), 53  
 dmenu\_executable setting, 62

## E

edit command, 41  
 edit\_account setting, 60  
 edit\_template setting, 60  
 encoding setting, 60  
 entropy, 10  
 exclude() (in module avendesora), 52

## F

find command, 42  
 find\_accounts() (avendesora.PasswordGenerator method), 63  
 Firefox browser, 27

## G

generated secrets, 9  
 get\_account() (avendesora.PasswordGenerator method), 64  
 get\_composite() (avendesora.Account class method), 64  
 get\_culprit() (avendesora.PasswordError method), 66  
 get\_fields() (avendesora.Account class method), 64  
 get\_message() (avendesora.PasswordError method), 66  
 get\_name() (avendesora.Account class method), 65  
 get\_passcode() (avendesora.Account class method), 65  
 get\_scalar() (avendesora.Account class method), 65  
 get\_username() (avendesora.Account class method), 65  
 get\_value() (avendesora.Account class method), 65  
 get\_values() (avendesora.Account class method), 66  
 gmail, 73  
 google, 73  
 Google Authenticator, 32  
 GPG (class in avendesora), 54  
 gpg\_armor setting, 61  
 gpg\_executable setting, 61  
 gpg\_home setting, 61  
 gpg\_ids setting, 61

## H

hashes file, 59  
 help command, 42  
 HEXDIGITS (avendesora attribute), 53  
 Hidden (class in avendesora), 54  
 Hide (class in avendesora), 53  
 highlight\_color setting, 61

## I

identity command, 42  
 initial configuration, 12  
 initialize command, 42  
 installing, 11

## L

label\_color setting, 60  
 LETTERS (avendesora attribute), 53  
 log command, 43  
 log file, 43, 59  
 log\_file setting, 59  
 login command, 41  
 LOWERCASE (avendesora attribute), 53

## M

misdirection, 36  
 MixedPassword (class in avendesora), 49

## N

networth example, 70  
 new command, 43  
 None, 46

## O

One-time passwords, 32  
 OTP, 32  
 OTP (class in avendesora), 51

## P

Passphrase (class in avendesora), 47  
 Password (class in avendesora), 46  
 PasswordError, 66  
 PasswordGenerator (class in avendesora), 63  
 PasswordRecipe (class in avendesora), 50  
 phishing, 26  
 phonetic alphabet, 38  
 phonetic command, 43  
 PIN (class in avendesora), 48  
 postmortem letter example, 69  
 previous\_archive\_file setting, 59  
 PRINTABLE (avendesora attribute), 53  
 PUNCTUATION (avendesora attribute), 53

**Q**

Question (class in avendesora), 48  
 questions, 29, 72  
 qutebrowser, 27

**R**

RecognizeAll, 28  
 RecognizeAll (class in avendesora), 54  
 RecognizeAny, 28  
 RecognizeAny (class in avendesora), 55  
 RecognizeCWD (class in avendesora), 56  
 RecognizeEnvVar (class in avendesora), 57  
 RecognizeFile, 29  
 RecognizeFile (class in avendesora), 58  
 RecognizeHost (class in avendesora), 56  
 RecognizeNetwork (class in avendesora), 57  
 RecognizeTitle, 26  
 RecognizeTitle (class in avendesora), 55  
 RecognizeURL, 27  
 RecognizeURL (class in avendesora), 55  
 RecognizeUser (class in avendesora), 56  
 render() (avendesora.PasswordError method), 66  
 report() (avendesora.PasswordError method), 66  
 report() (avendesora.SecretExhausted method), 52  
 reveal command, 44

**S**

Script (class in avendesora), 58  
 scripts, 33  
 search command, 44  
 search\_accounts() (avendesora.PasswordGenerator method), 64  
 second factor, 32  
 SecretExhausted, 52  
 security questions, 29, 72  
 selection\_utility setting, 61  
 settings, 59  
 shell account, 14  
 shell command recognition, 78  
 SHIFTED (avendesora attribute), 53  
 short cuts, 16  
 ssh key example, 67  
 stealth accounts, 35  
 stealth\_accounts file, 59  
 swarm accounts, 77  
 SYMBOLS (avendesora attribute), 53

**T**

terminate() (avendesora.PasswordError method), 67  
 terminate() (avendesora.SecretExhausted method), 52  
 two page authentication, 73  
 typing, reducing, 16

**U**

upgrading, 79  
 UPPERCASE (avendesora attribute), 53  
 use\_pager setting, 61

**V**

value command, 44  
 values command, 45  
 verbose setting, 61

**W**

website account, 18  
 WHITESPACE (avendesora attribute), 53  
 window manager, 13  
 wireless router, 74  
 with\_traceback() (avendesora.PasswordError method), 67

**X**

xdotool\_executable setting, 61  
 xsel\_executable setting, 62