$AVE IT - User Guide

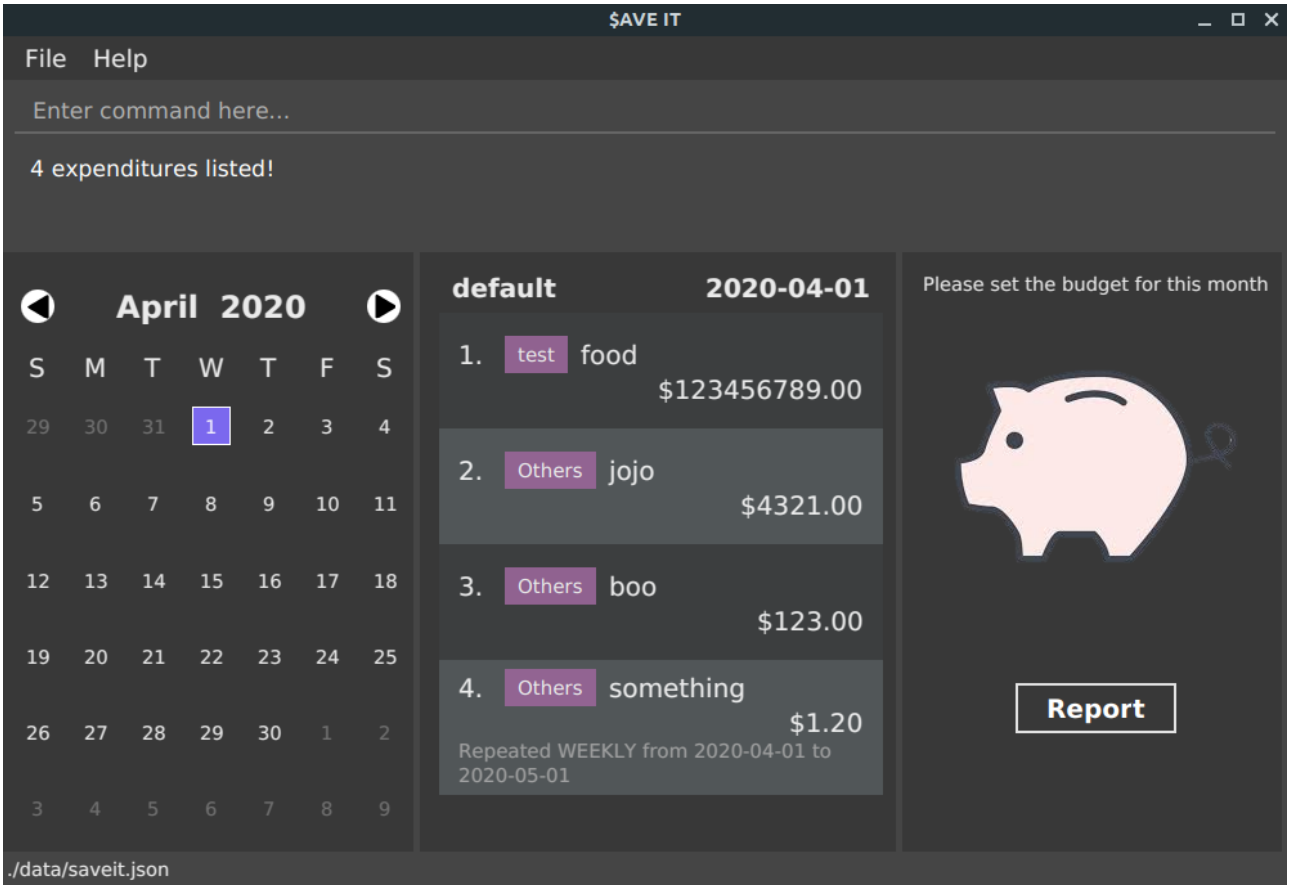
By: `Team T10-3` Since: `Mar 2020` Licence: `MIT`

1. Introduction

$AVE IT is a **desktop budget management application**. We are here to assist tertiary students who have needs in financial budgeting and management. Tertiary students who have to manage finance accounts from different sources such as school project and personal spending will find ease in using our application.

2. Quick Start

1. Ensure you have Java `11` or above installed in your Computer.
2. Download the latest `$AVE IT.jar` here.
3. Copy the file to the folder you want to use as the home folder for $AVE IT application.
4. Double-click the file to start the app. The GUI should appear in a few seconds.



5. Type the command in the command box and press `Enter` to execute it.
   e.g. typing `help` and pressing `Enter` will open the help window.
6. Some example commands you can try:
   - `exp add -i chicken rice -a 4.50`: adds new expenditure and information.
   - `exp delete 3`: deletes the 3rd expenditure shown in the current list.
   - `exit` : exits the app.
7. You can refer to Main Commands for details of each command.

3. Features

`$AVE IT` consists of features such as disjoint account, expenditure management, recurring expenditure, budgeting, report. Additional features are calendar, autocomplete and auto-saving of data.

3.1. Disjoint Accounts (Jiang Jiahui)

This feature aims to help you better organise your expenditures by allowing you to separate them into disjoint accounts.

You can add any many accounts as you like:

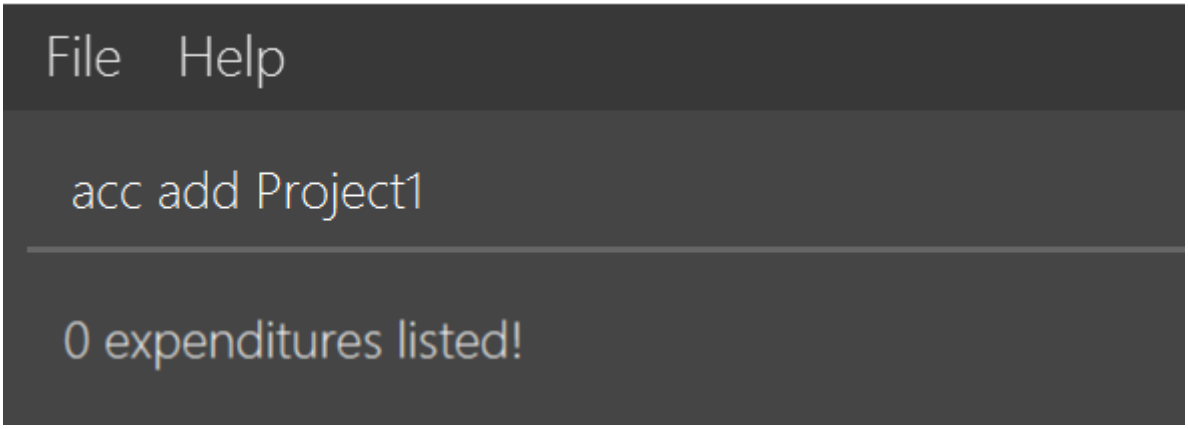


Figure 1. Add account

View your list of accounts using the command `acc list`:

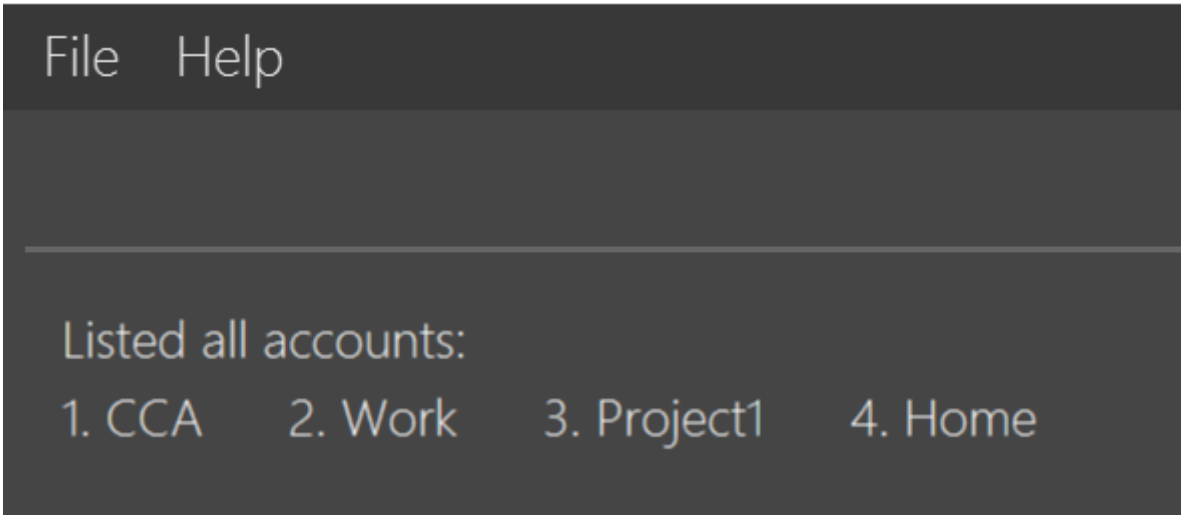


Figure 2. List all accounts

Switch to a different account using the command, `acc checkout ACCOUNT`:

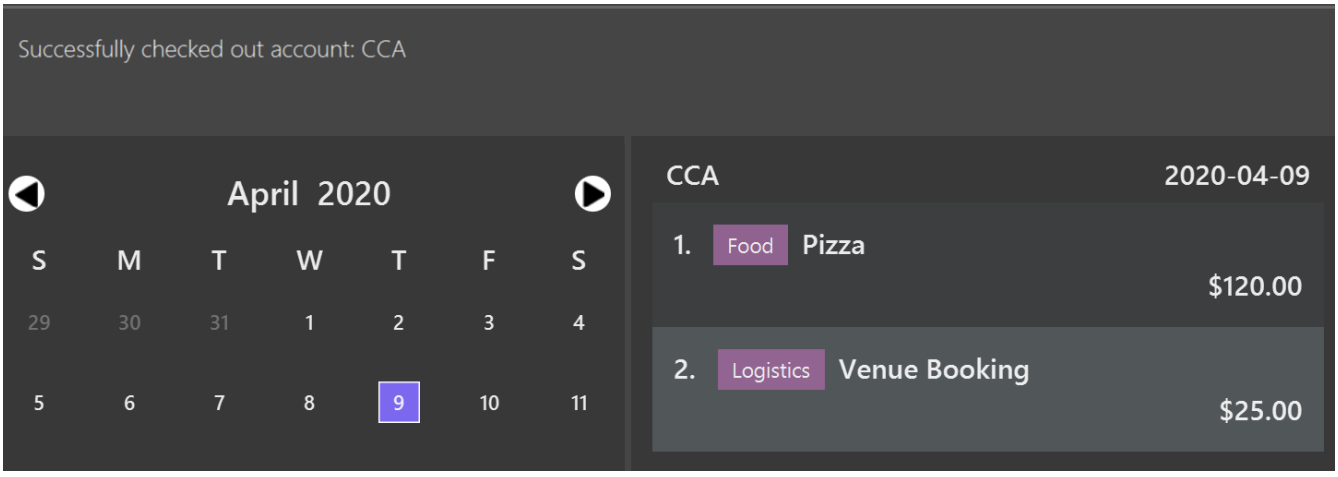


Figure 3. Switch account to `CCA`

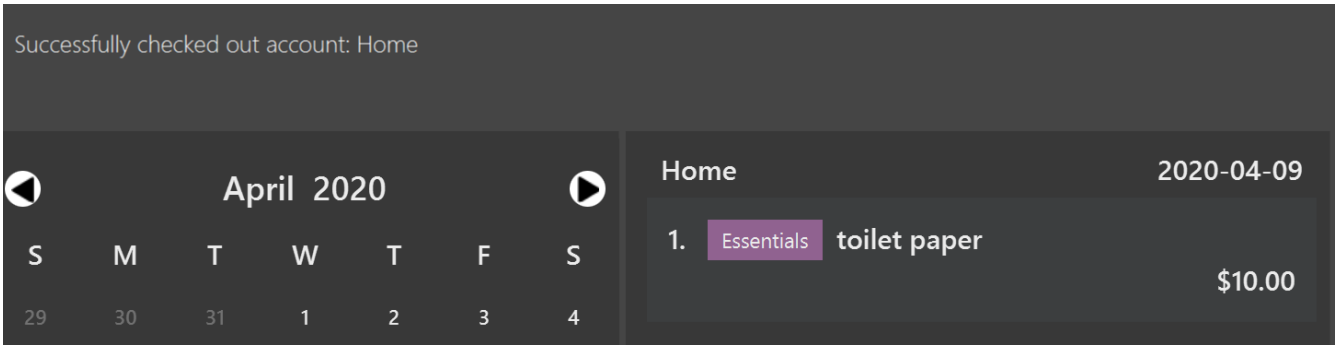


Figure 4. Switch account to `Home`

Refer to Account Commands for more details on commands, including renaming, deletion and clearing of data.

3.2. Expenditures (Jiang Jiahui)

This feature forms the basis of our application. Use it to track your daily expenses!

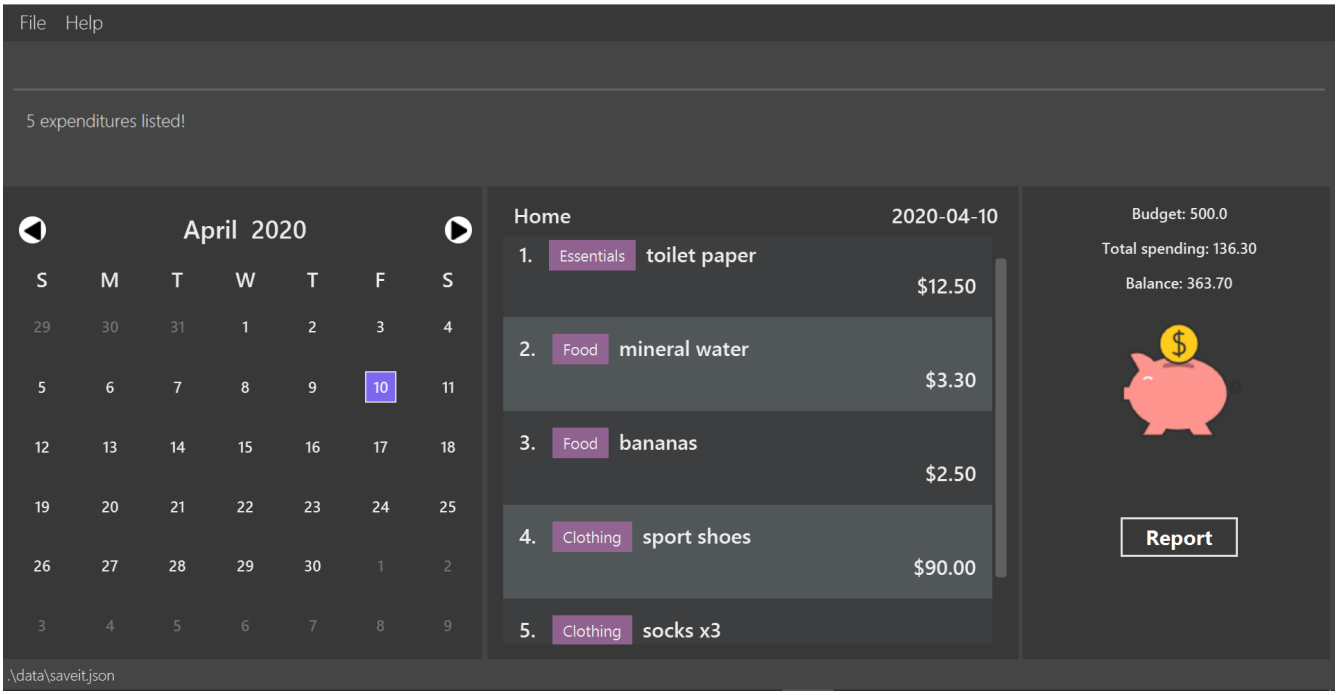


Figure 5. Example list of expenditures

Refer to Expenditure Commands for more details on how to add, edit, and delete expenditures.

3.3. Repeat (Zheng Shaopeng)

The *Repeat* feature allows you to add recurring *expenditures*.
You are able to create fixed expenditures records which will be recurring daily, weekly, monthly or annually. As shown in the diagram below, a `repeat` will have a different display as compared to `expenditure`. "Repeat details" includes the frequency of recurring, start date and end date are displayed.

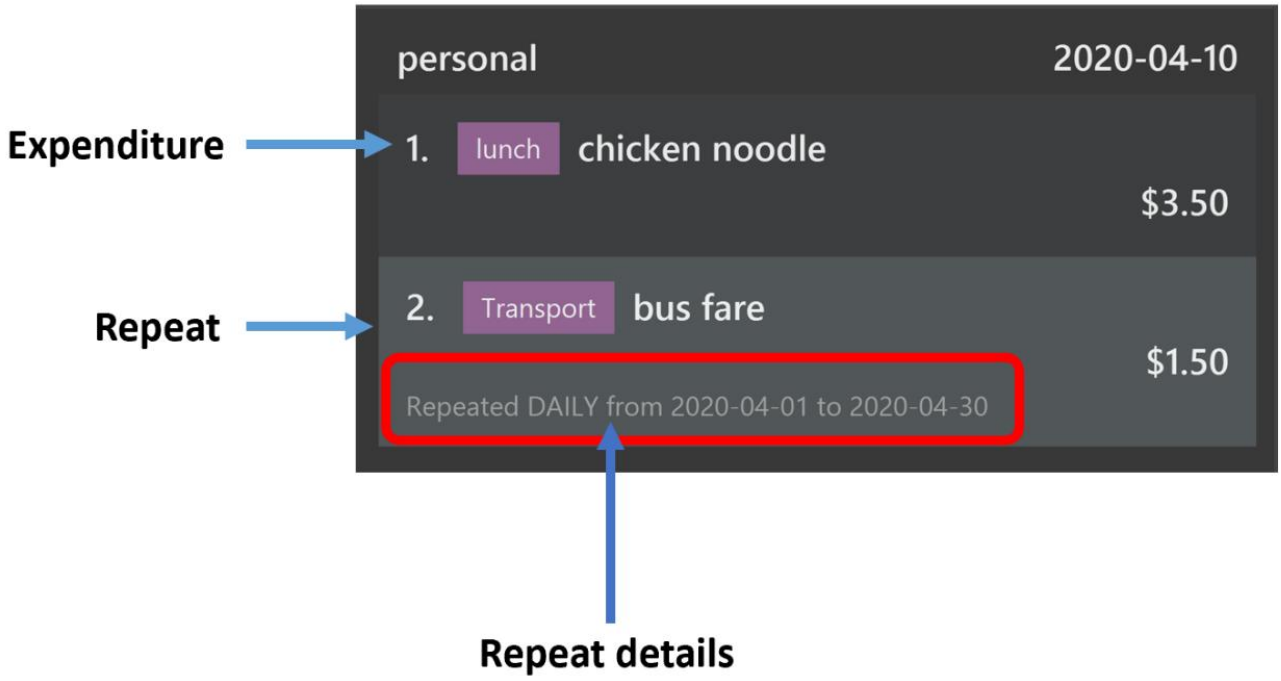


Figure 6. Display difference between `Expenditure` and `Repeat`

3.3.1. `Repeat` type commands

`Repeat` has very similar command as `Expenditure`, with just a few of values to be specified.
Here, we will be using `repeat add` command as an example, while other `repeat` commands works the same way.
Format: `repeat add -i INFO -a AMOUNT -sd START_DATE -ed END_DATE -p PERIOD [-t TAG]`

1. Key in the command into the command box. We will be using `repeat add -i bubble tea -a 3.50 -sd 2020-04-01 -ed 2020-04-30 -p weekly -t drink` as an example.

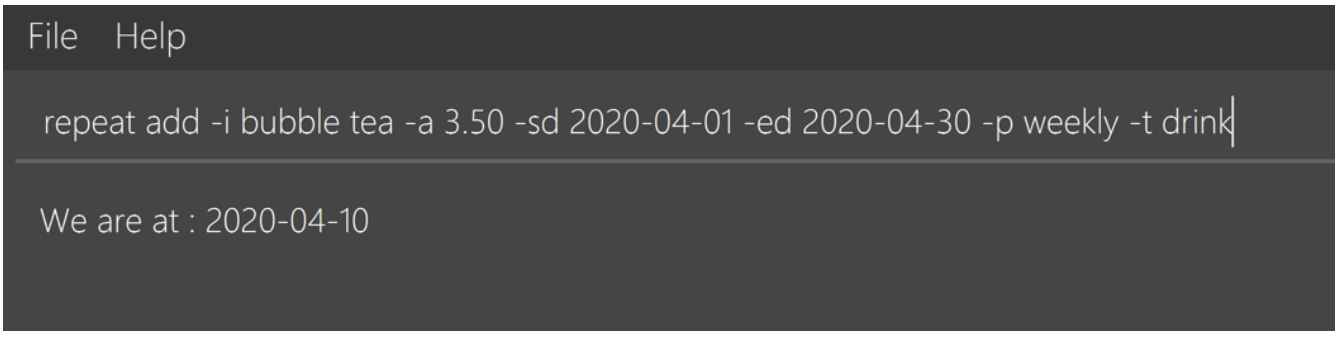


Figure 7. Input command

2. A response will be given to indicate that this recurring expenditure has been recorded.

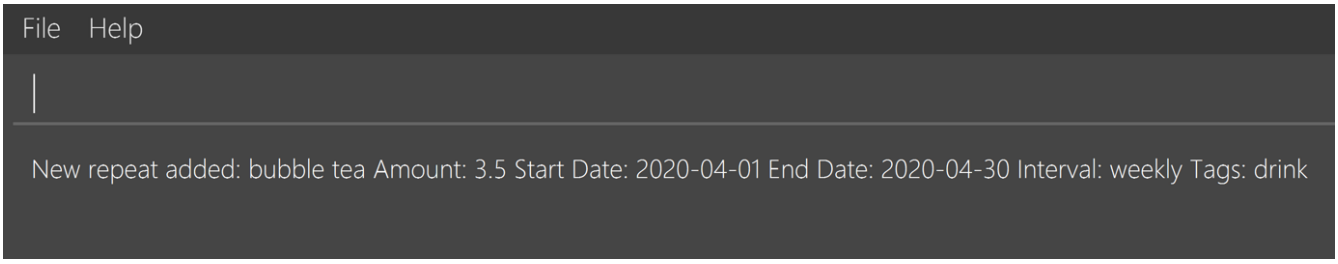


Figure 8. Response given

3. As the `START_DATE` is 2020-04-01 and the `PERIOD` is set to weekly, thus the first record will be at 2020-04-01 and last record will be on 2020-04-29.

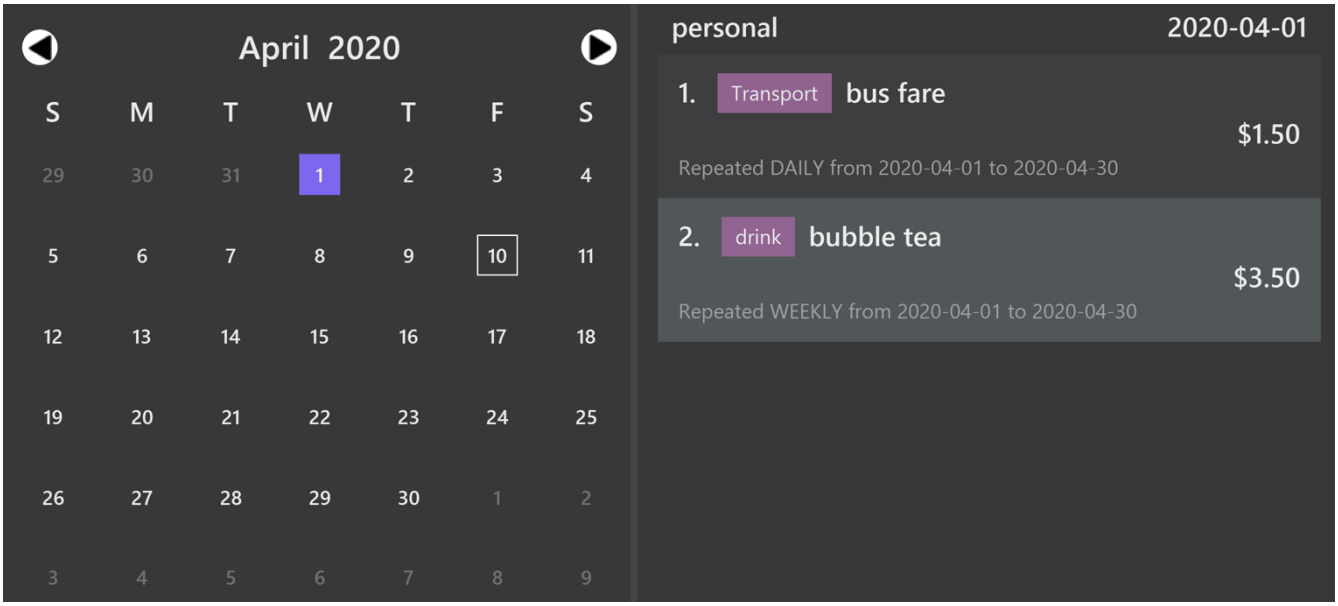


Figure 9. Added to 2020-04-01

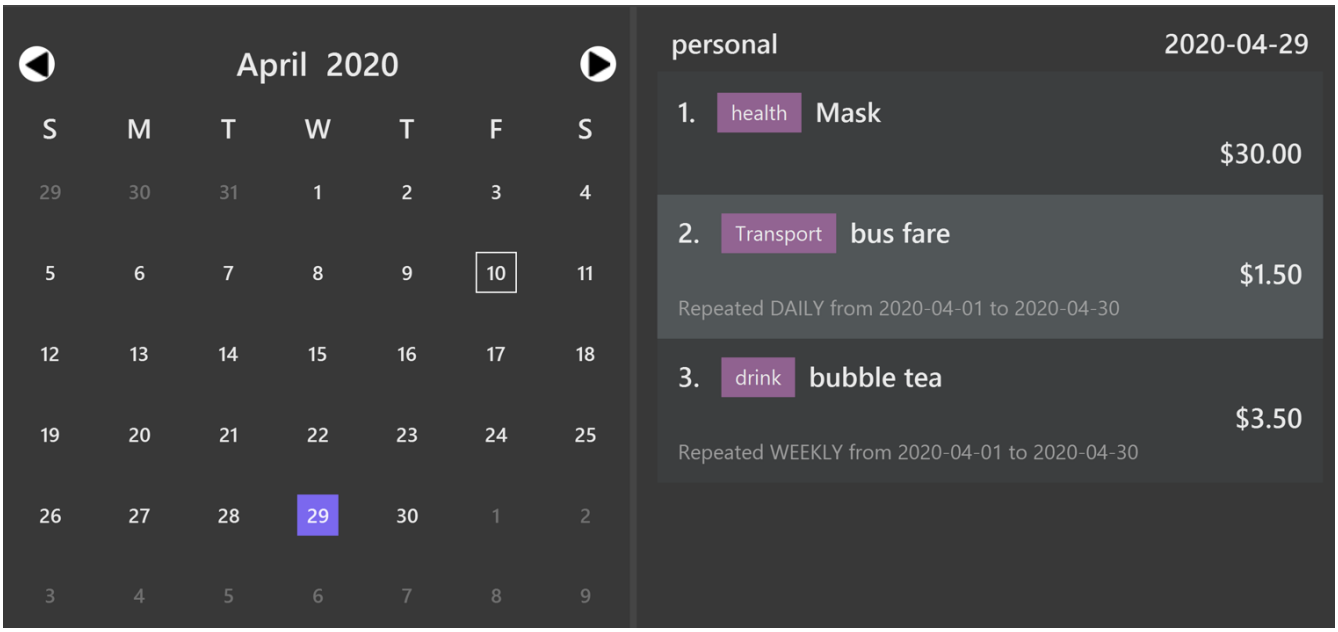


Figure 10. Added to 2020-04-29

NOTE

For `repeat add -i rental fee -a 300 -sd 2020-03-31 -ed 2021-03-31 -p monthly -t housing`, as the start date is `2020-03-31`, the next tentative date should be `2020-04-31` but this date is invalid. Hence, this expenditure record will be shown on `2020-04-30` instead. For May, it will be still shown on `2020-05-31`.
This applies to leap year too

Refer to Repeat Commands for more details on how to add, edit and delete `Repeat` type expenditures.

3.4. Budget (Lim Feng Yue)

The *Budget* feature allows you to set your budget monthly, and give you a better sense of whether you are on track to your financial goals. You can see your budget at the right panel of the application.

Initially, you will see this:

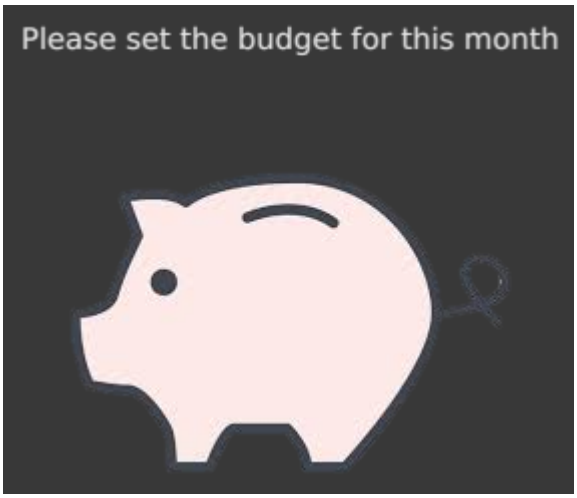


Figure 11. No budget set

It means that there is not budget set for the month.

In order to know which month `this month` in the above image refers to, you can look at the date in the middle panel.

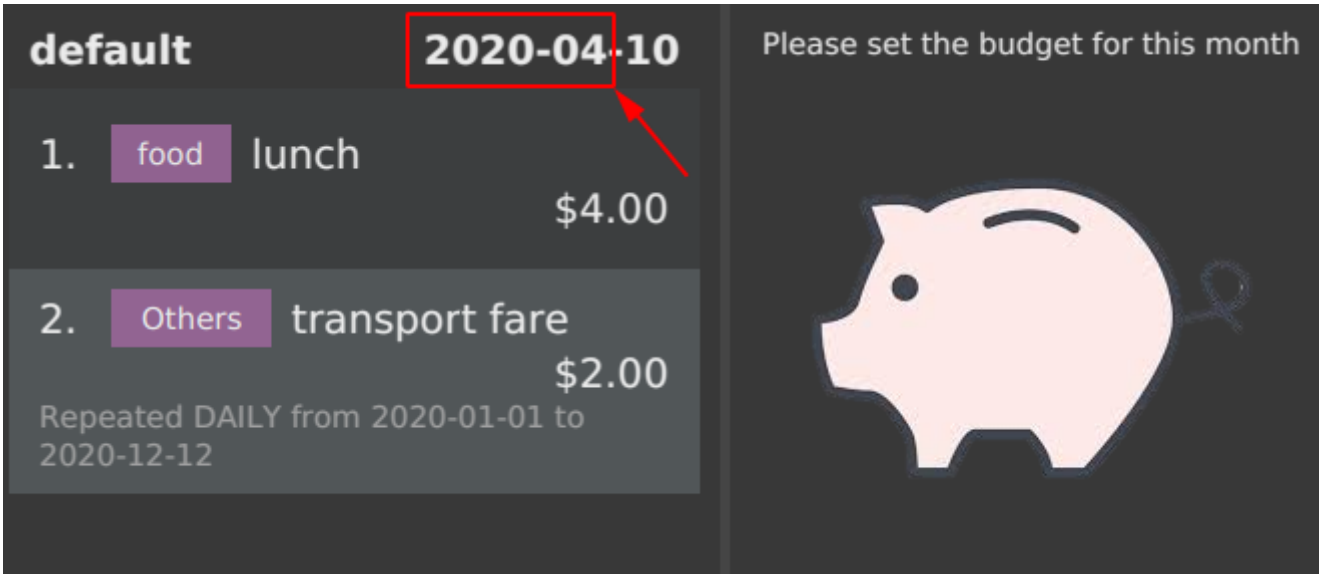


Figure 12. No budget set for April 2020

3.4.1. Setting a budget

To set a budget you can use the `setbudget` Command. Shown below is an example:

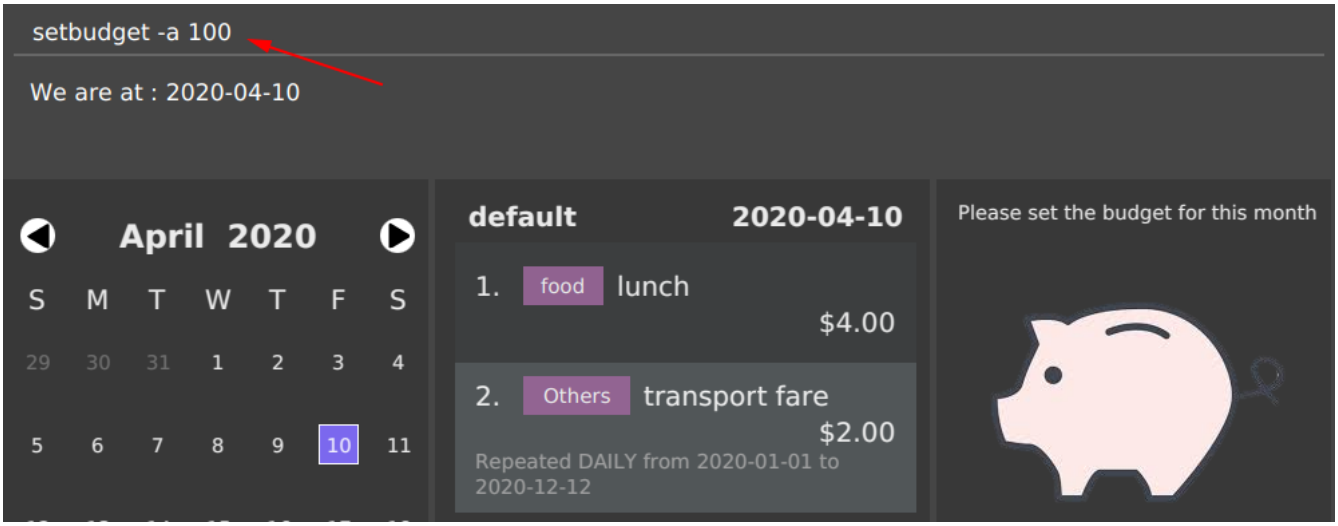


Figure 13. Setting this month's budget to $100

After setting a budget, you will see that the information on the right panel has changed:

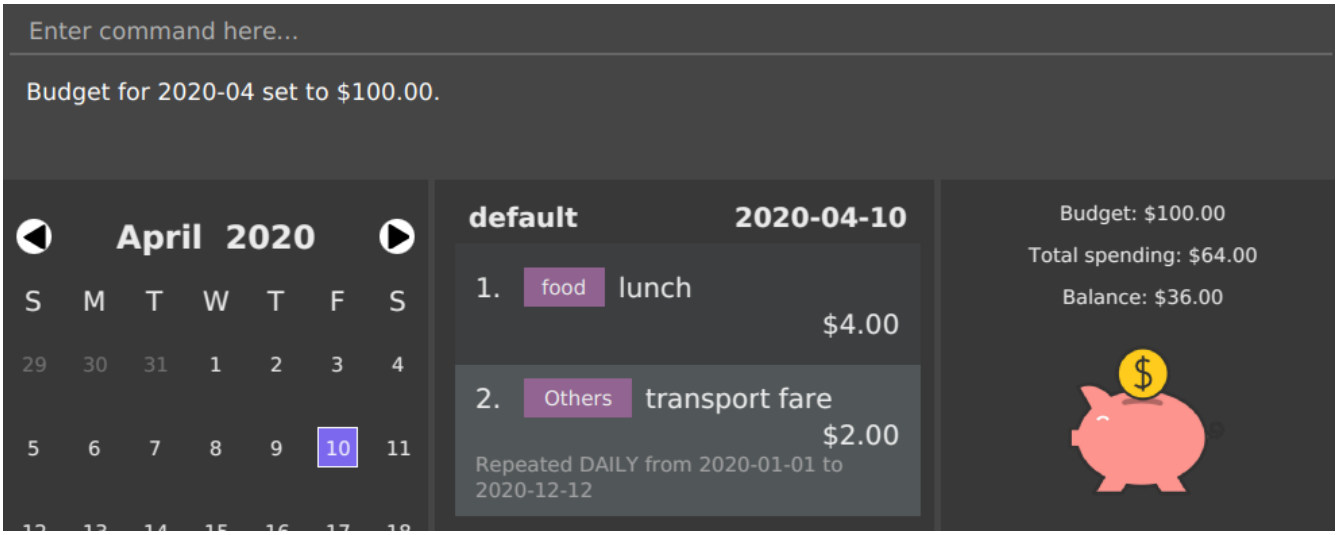


Figure 14. After setting a budget for April 2020

If your budget has been met, you will see the piggy bank from above. If your budget is not met, you will see something like this:

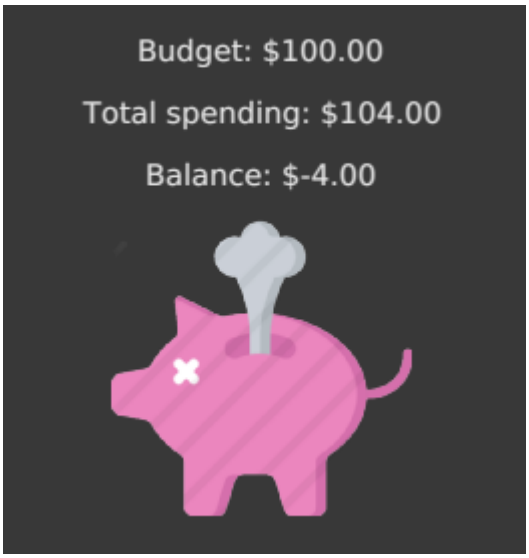


Figure 15. Budget not met

NOTE	To meet your budget, your balance have to be positive.

3.4.2. Viewing budget from another month

To view budget from another month, you can use the `go` command, or the Calendar feature to view any date of the month.

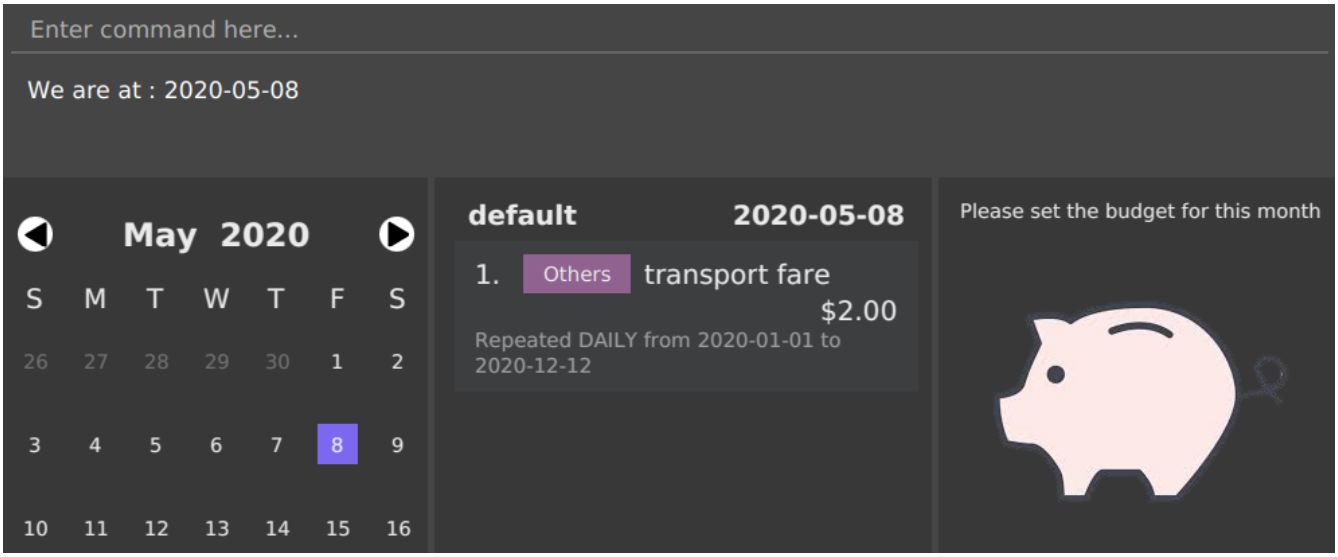


Figure 16. View May 2020 budget

For example, going to *8 May 2020* allows you to view the budget set for *May 2020*.

3.5. Report (Ng Xin Pei)

3.5.1. Function

The *Report* feature allows you to have a quick overview of your expenditures. It can tell you your total spending within a period of time and have your spending categorised either by month or tag. You can have also have a permanent copy of the report, through the report printing and export capabilities.

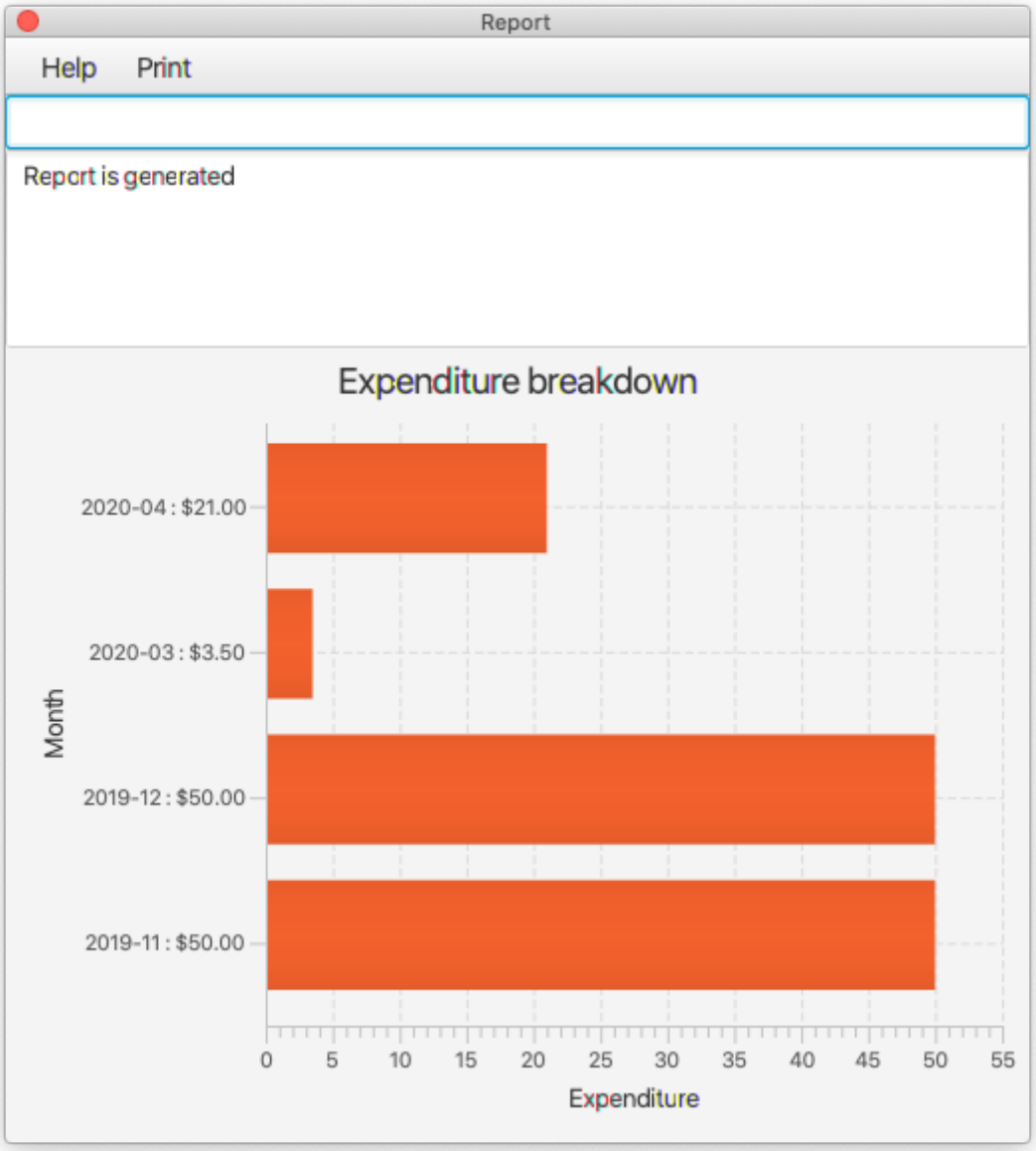


Figure 17. Bar chart report

This is an example of the report in bar chart format with the expenditures grouped by months. The horizontal line represents the expenditures whereas the vertical line shows the month followed by the total spending in that month. Alternatively, you can also group the expenditure by tags and read off the bar chart in a similar fashion.

If you are a visual person, you can consider using the pie chart format instead!

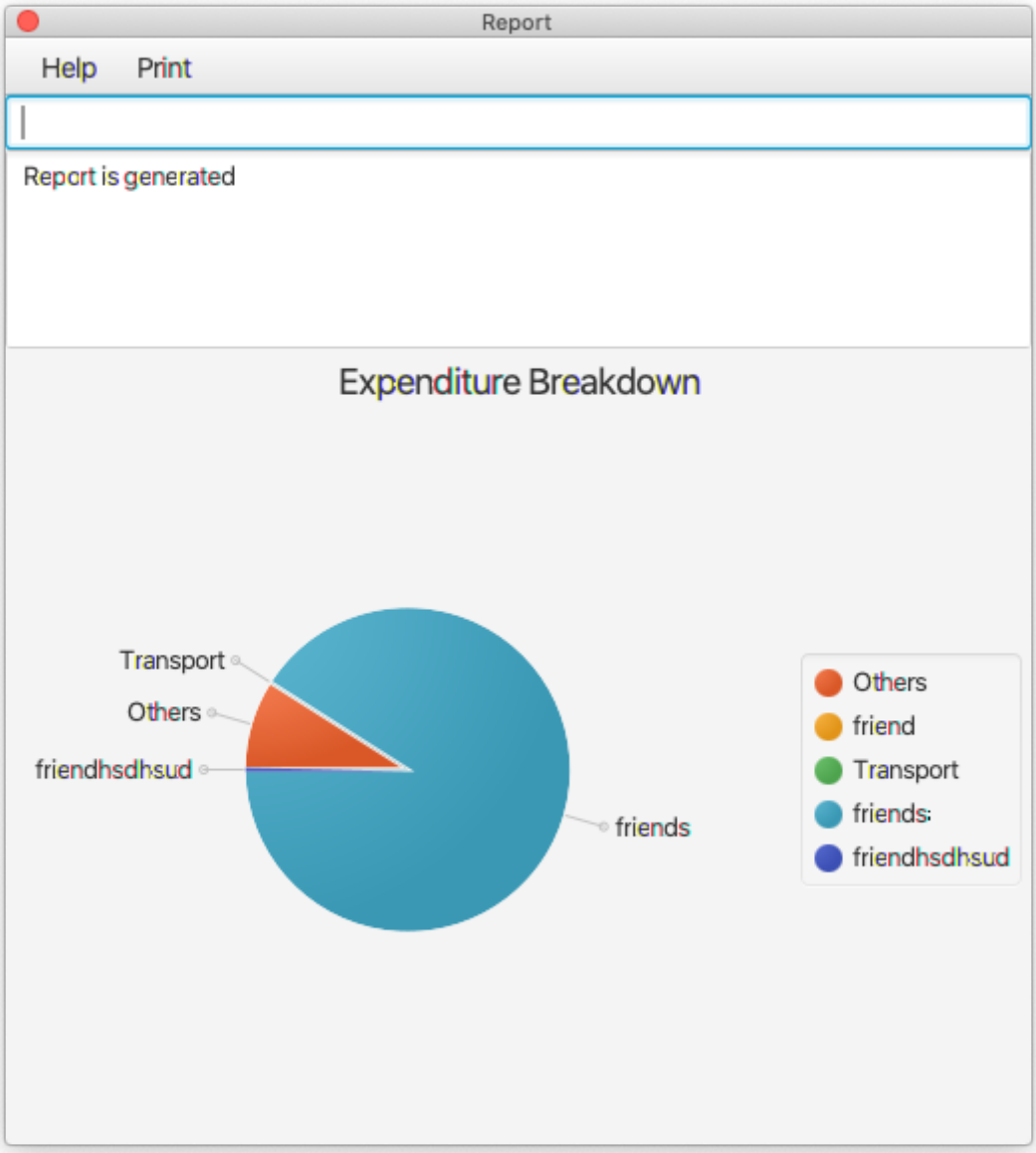


Figure 18. Pie chart report

This is an example of report in pie chart format with the expenditures grouped by tags.

If you are interested to find out more, you can explore Report Command to find out how to generate these graphs.

3.6. Calendar (Zheng Shaopeng)

The *calendar* feature aims to ease users' navigation experience.

1. Users are able to traverse between different dates by clicking on the calendar.
   Or,
2. Users are able to make use of go command to navigate to another date.

With reference to the diagram below, **Header** displays the year and month which the calendar is displaying. **Box 1** displays the day which you are looking at while **Box 2** refers to the today's date.

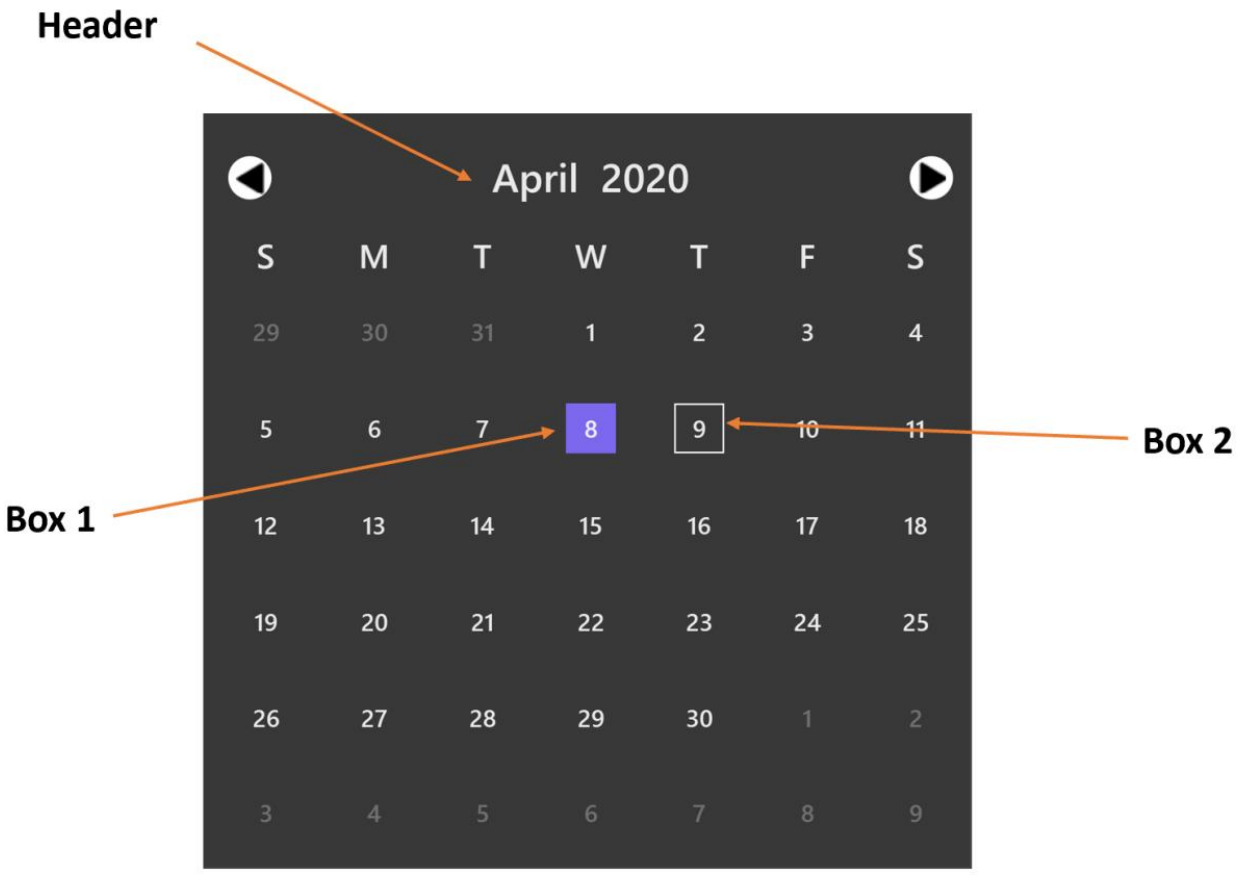


Figure 19. Calendar View

3.6.1. Navigate using calendar

It may be troublesome and inconvenient for you to keep on making use of the command line input to navigate between the dates, especially when the date you want to view is just one or two days before.
In this situation, you can interact with the `calendar` and navigate with a simple click.

For example, we are now viewing expenditure records which are on `2020-04-09`. And we wish to view previous day's expenditure record. We can simply click on the date on the calendar. Below are two diagrams which will show the operation.

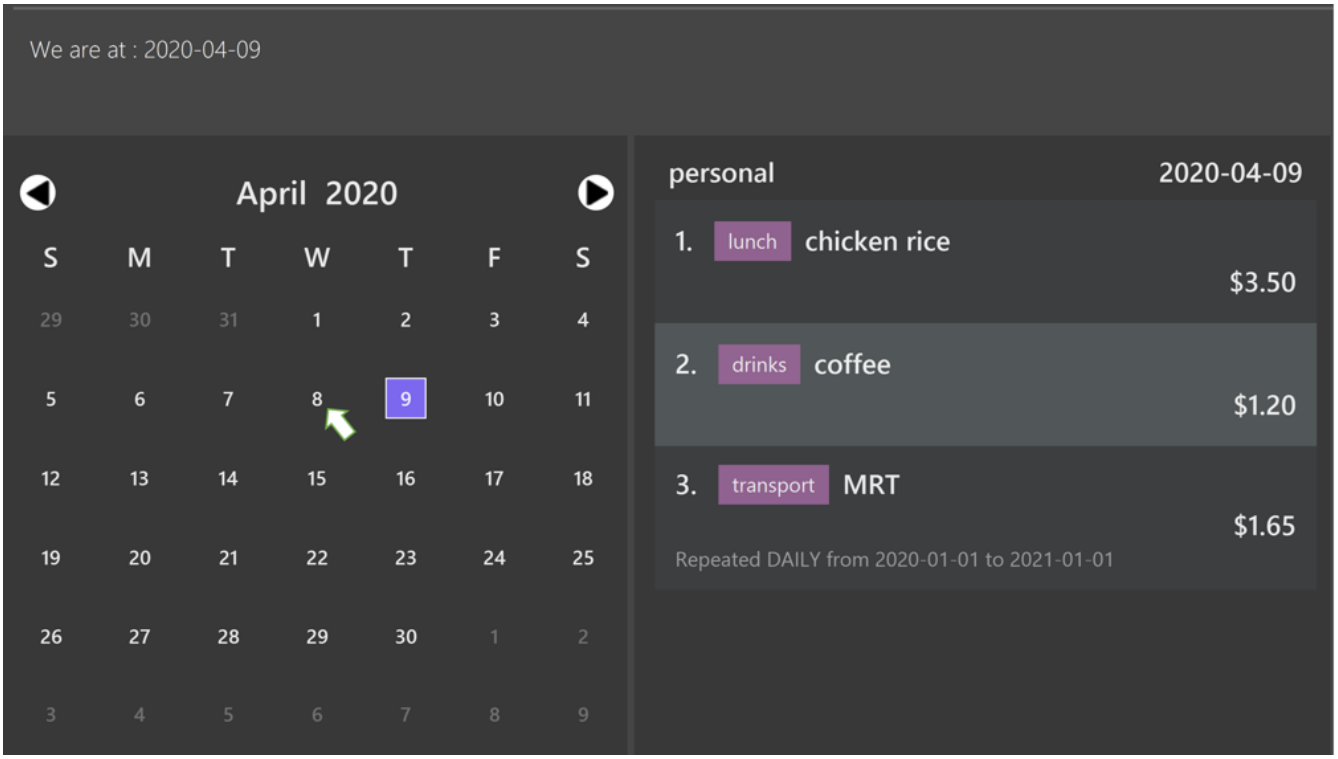


Figure 20. Before click (9 April)

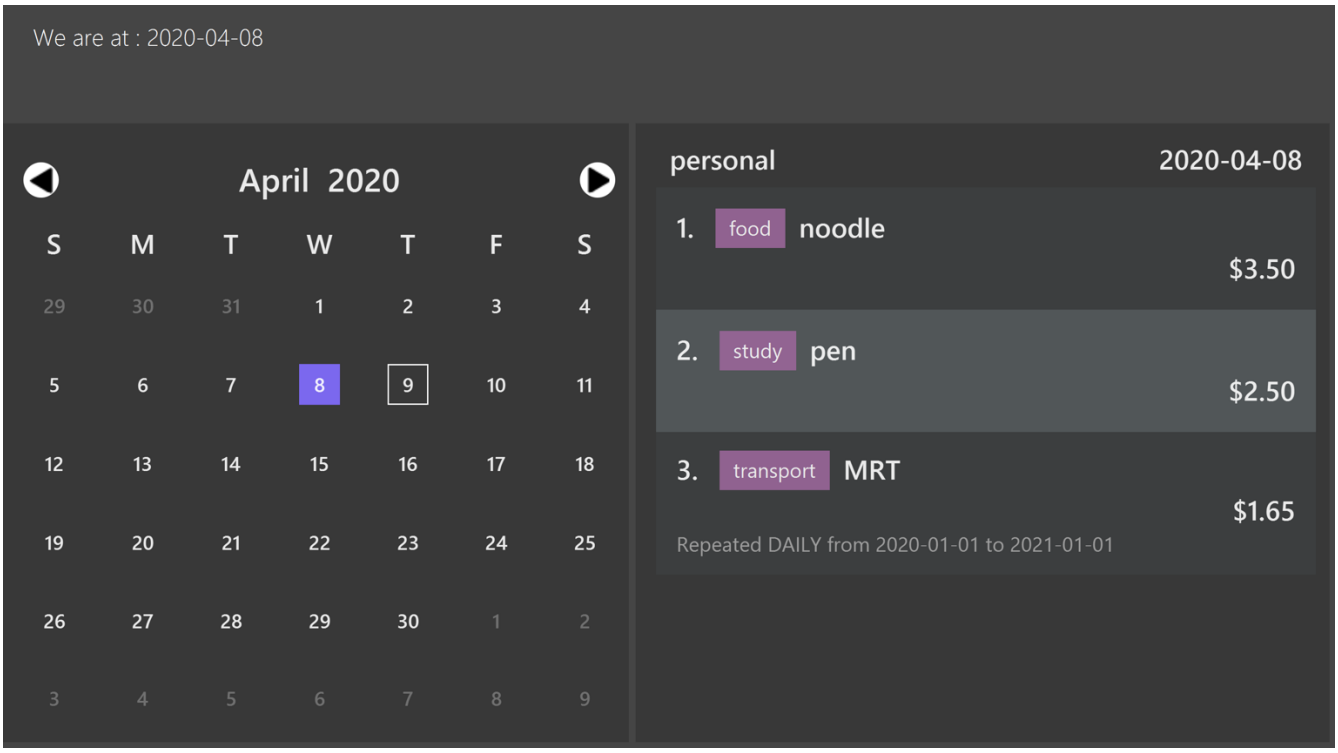


Figure 21. After click (8 April)

3.6.2. Navigate with `go` command

The developer team has taken into account the case when users wish to navigate to another date which is way before or after. Thus, we have integrate the `calendar` with `go` command.

For example, we are now viewing expenditure records which are on `2020-04-09` and we wish to navigate to `2019-04-09` through `go` command.

1. We need to input `go 2019-04-09` into the command box which is shown in the diagram below.

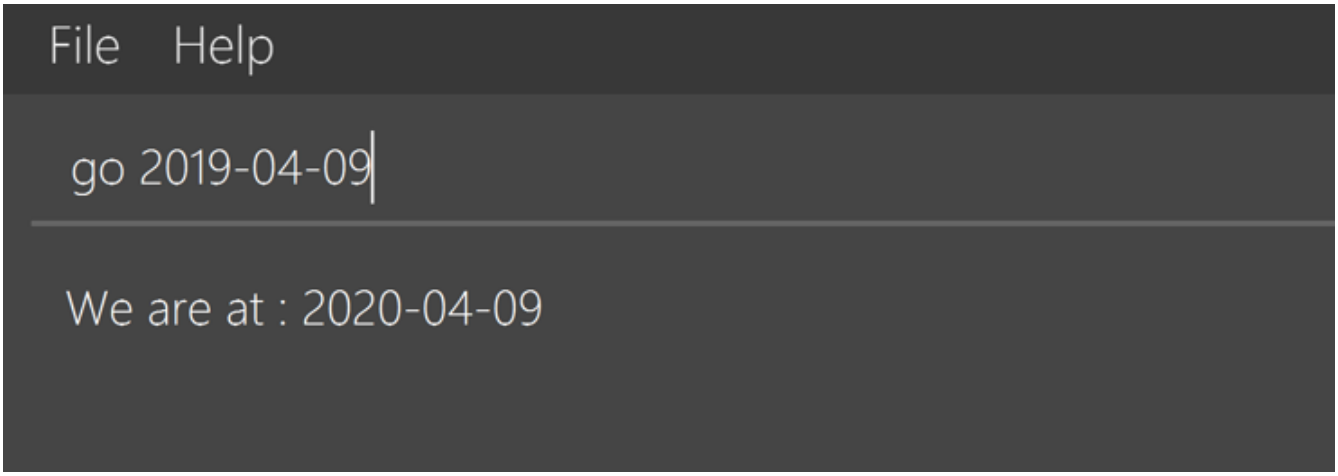


Figure 22. `go` command example

2. A response will be given to indicate that the date has changed.

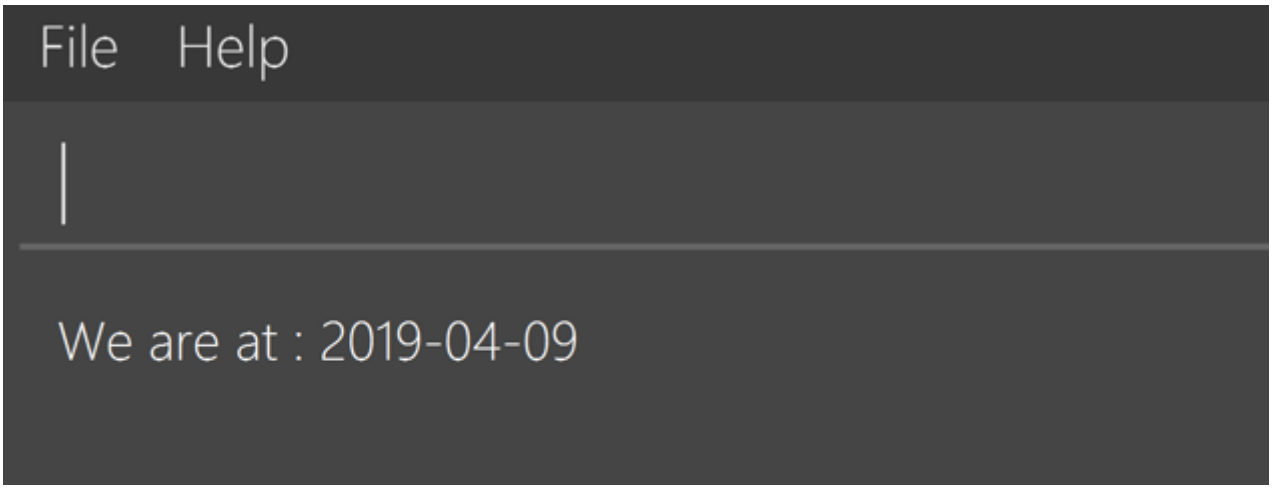


Figure 23. `go` command response

3. The `calendar` and the expenditure records will update accordingly.

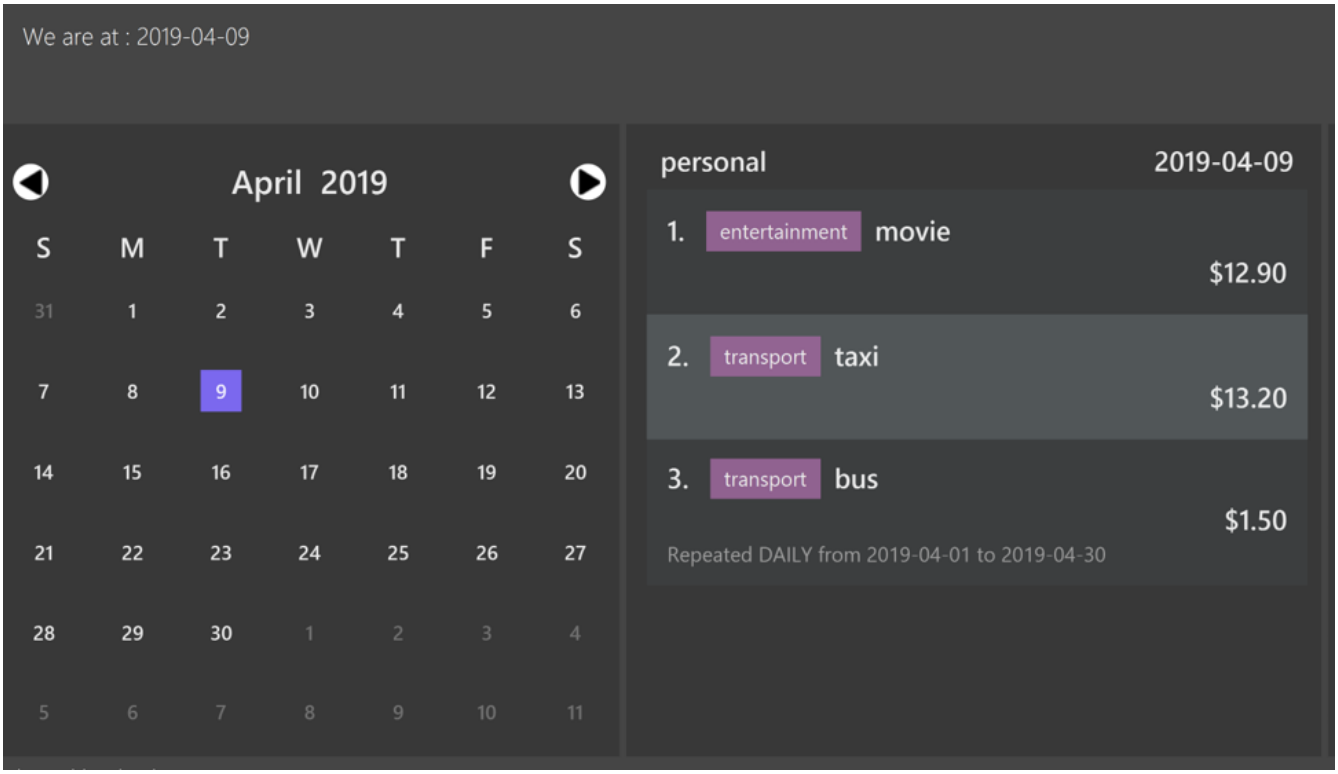


Figure 24. `calendar` and expenditure records are updated accordingly.

3.7. Autocomplete (Lim Feng Yue)

The *Autocomplete* feature allows you to complete the basic commands of the application. It matches the what you type into the command box and tries to complete the command.

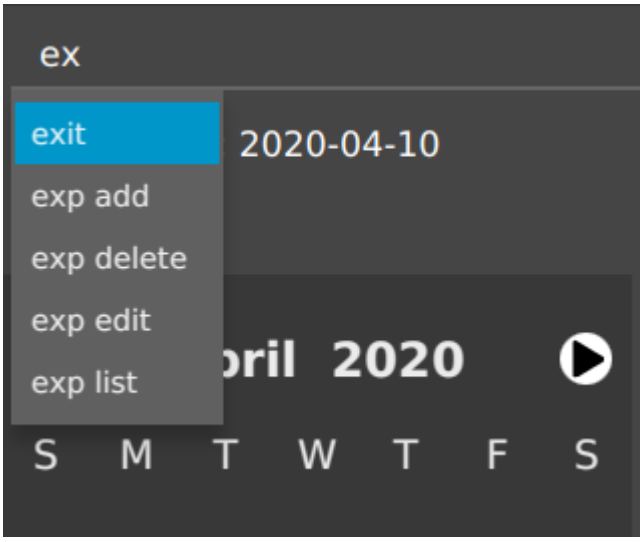


Figure 25. Autocomplete feature

TIP	If no dropdown appear, it means that either: the text does not match any commands, or the dropdown is hidden after clicking away.

You can use the `Up` or `Down` arrow keys to navigate the dropdown, then press `Enter` to select the command you want to complete.

NOTE	The existence of the dropdown menu may require you to press `Enter` 2 times before the command is executed, due to how the autocompletion works.
NOTE	When pressing `Down` arrow key does not allow you to select the commands, it means that the text box is not focused. Clear all texts and try again.

3.8. Saving the data

$AVE IT data is stored in the hard disk automatically after any command that changes the data. There is no need to save manually.

4. Main Commands

Command Format:

Let's use a command format,
`exp add -i INFO -a AMOUNT [-t TAG] [-d DATE]`
as an example.

- The first word is the **operation category** of the command, in this case, `exp` belongs to the expenditure operations.
- There may be a second word which is the specific command for the operation category, in this case, `add`.
- Words starting with `-` e.g. `-i`, `-a` and `-t` are the **option prefix**. Not all commands have options, an example is deleting an expenditure, `exp delete 1`.

NOTE	Refer to Section 7.1, "Prefix Notation" for what each prefix means. Only the last valid prefix will be taken into account. E.g. `exp add -i chicken rice -a 3.5 -a 4.0`. The new expenditure record will have an amount fo 4.0 instead of 3.5. **This applies to all other command**.

- Words in caps are the **parameters** to be supplied by the user. `INFO`, `AMOUNT` and `TAG` are parameters representing information, amount and tag of the expenditure respectively. It can be used as such, `exp add -i chicken rice -a 4.50 -t food`.
- Items in square brackets are optional, that means in this example `exp add -i chicken rice -a 4.50` is also valid.
- Parameters can be in any order e.g. if the command specifies `-i INFO -a AMOUNT`, `-a AMOUNT -i INFO` is also acceptable.
- However, if the parameter is not prefixed by any option prefixes, the order of the parameter matters.

4.1. Viewing help: `help`

Views the help message.
Format: `help [COMMAND]`

- When the optional `COMMAND` is stated, more information on the category of commands will be shown.

TIP The available `COMMAND` will be shown when entering `help`.

Examples:

- `help`
- `help exp`
- `help go`

4.2. Going to a specific date: `go`

Lists the expenditures from a particular date.
Format: `go DATE`

- `DATE` is a date in the format of `YYYY-MM-DD`, or year month day.

TIP Use `go today` to go to today's date.

Examples:

- `go 2020-05-05`
- `go today`

4.3. Account commands

4.3.1. Adding an account: `acc add`

Adds a new disjoint account to $AVE IT.
Format: `acc add ACCOUNT`

- `ACCOUNT` is the account name. It must be less than 26 characters and must not contain any space in between.
- `ACCOUNT` must only contain letters and numbers, and it is also case-sensitive.
- The default account is called `default`, it is automatically created when you first use the application.

Examples:

- `acc add school`
- `acc add SpendLess1`

> **TIP** The application does not automatically switch to the new account. Use `acc checkout ACCOUNT` to do so.

4.3.2. Listing all accounts: `acc list`

List all disjoint accounts in $AVE IT.
Format: `acc list`

4.3.3. Renaming an account: `acc rename`

Renames an existing account in $AVE IT.
Format: `acc rename [OLD_NAME] NEW_NAME`

- `OLD_NAME` is the name of an already existing account.
- if `OLD_NAME` is not specified, the current account will be renamed.
- `NEW_NAME` should not be the name of an existing account.
- See Section 4.3.1, “Adding an account: `acc add`” for account naming constraints.

Example:

- `acc rename project2`
  Renames the current account to `project2`.
- `acc rename school uni`
  Renames the account `school` to `uni`.

4.3.4. Changing the current account: `acc checkout`

Changes the account view to another existing account.
Format: `acc checkout ACCOUNT`

> **TIP** Use `acc list` to view the available accounts.

NOTE Did you know git's `git checkout` command is our inspiration for `acc checkout`?

Example:

- `acc checkout cca`
  Sets `cca` to be the current account.

4.3.5. Clearing account information: `acc clear`

Reset the current account to a clean state.
Format: `acc clear`

WARNING All information in the current account will be deleted, use with care.

4.3.6. Deleting an account: `acc delete`

Deletes an existing account in $AVE IT.
Format: `acc delete ACCOUNT`

- `ACCOUNT` is the name of the account that will be deleted.

WARNING The account and all the information in it will be deleted, use with care.

Example:

- `acc delete school`
  Deletes the account named `school`.

4.4. Expenditure commands

4.4.1. Adding an expenditure: `exp add`

Adds an expenditure to the current account in $AVE IT.
Format: `exp add -i INFO -a AMOUNT [-t TAG] [-d DATE]`

- `INFO` contains a description of what you spent on.
- `AMOUNT` is how much you have spent (in dollars). It must be positive up to 2 decimal point and less than 1 billion.
- `TAG` is the category of the expenditure. The default tag is `Others`. Each expenditure belongs to only one tag.
- `DATE` is the date on which the expenditure happened. When not specified, will default to the active date (as shown in the middle section, or by the coloured date in the calendar)

Examples:

- `exp add -i chicken rice -a 4.50`
- `exp add -i chicken rice -a 4.50 -t food -d 2020-01-01`

4.4.2. Editing an expenditure : `exp edit`

Edits an existing expenditure in the current account in $AVE IT.
Format: `exp edit INDEX [-i INFO] [-a AMOUNT] [-t TAG] [-d DATE]`

- Edits the expenditure with the specified `INDEX`.
  `INDEX` refers to the index number assigned to each expenditure based on the order.
- At least one of the optional fields must be provided.
- Existing values will be updated to the input values.
- The expenditure records will be auto sorted again.
- Refer to Section 4.4.1, "Adding an expenditure: `exp add`" for what the options represent.

Examples:

- `exp edit 1 -i veg rice`
  Updates the info of expenditure with index `1` to `veg rice`.
- `exp edit 3 -a 3.23 -t transport -d 2020-02-02`
  Updates the amount, tag & date attributes to the new values.

4.4.3. Deleting an expenditure: `exp delete`

Deletes an existing expenditure in the current $AVE IT account.
Format: `exp delete INDEX`

- Deletes the expenditure with the specified `INDEX`.
  `INDEX` refers to the index number assigned to each expenditure based on the order.

Example:

- `exp delete 4`
  Deletes the expenditure with index `4`.

4.4.4. Listing expenditures: `exp list`

Lists the expenditures for the current active date in the current account.
Format: `exp list`

TIP Use this command after `find` to return back to normal daily view.

4.5. Repeat commands

4.5.1. Adding a repeated expenditure: `repeat add`

Adds an expenditure that can be repeated.
Format: `repeat add -i INFO -a AMOUNT -sd START_DATE -ed END_DATE -p PERIOD [-t TAG]`
Do note that duplicate `repeat` records are allowed.

- `INFO` is the information of the expenditure.
- `AMOUNT` is the amount of the expenditure (in dollars). It must be positive up to 2 decimal point and less than 1 billion.
- `START_DATE` is the date in which the expenditure will start recurring from.
- `END_DATE` is the date in which the expenditure will no longer recur.
- `PERIOD` is the interval the expenditure will repeat. Valid intervals are `daily`, `weekly`, `monthly` and `annually`.
- `TAG` is the category of the expenditure. The default category is `Others`. You can only specify a category.

Examples:

- `repeat add -i transport fee -a 100 -sd 2020-01-01 -ed 2020-12-30 -p monthly`
  Adds a repeated expenditure called `transport fee` which will be counted for $100 monthly on the first day of the month for the year of 2020.
- `repeat add -i transport fee -a 100 -sd 2020-01-01 -ed 2020-12-30 -p monthly -t transport`

4.5.2. Editing a repeated expenditure: `repeat edit`

Edits an expenditure that can be repeated.
Format: `repeat edit INDEX [-i INFO] [-a AMOUNT] [-sd START_DATE] [-ed END_DATE] [-p PERIOD] [-t TAG]`

- Edits the repeated expenditure with the specified `INDEX`.
  `INDEX` refers to the index number assigned to each expenditure based on the order.
- At least one of the optional fields must be provided.
- Existing values will be updated to the input values.
- The records will be auto sorted again.
- Refer to Section 4.5.1, “Adding a repeated expenditure: `repeat add`” for what the options represent.

Examples:

- `repeat edit 2 -i concession`

Edits the information of the repeated expenditure with index `2` to `concession`.

- `repeat edit 3 -p weekly -ed 2020-07-02`
  Edits the interval for the repeated expenditure with index `3` to weekly and ends at 2 July 2020.
- `repeat edit 3 -ed 2021-07-3" Edit the end date for the repeated expenditure, this allows user to extend or shorten the whole duration.

4.5.3. Deleting a repeated expenditure: `repeat delete`

Deletes an expenditure that can be repeated.
Format: `repeat delete INDEX`

> - Deletes the repeated expenditure with the specified `INDEX`.
> `INDEX` refers to the index number assigned to each expenditure based on the order.

Example:

- `repeat delete 4` Deletes the repeated expenditure with index `4`.

4.6. Budget setting command: `setbudget`

Sets the budget amount in an account for the specified month.
Format: `setbudget -a AMOUNT [-ym YEAR_MONTH]`

> - `AMOUNT` is the amount of the budget (in dollars). It must be positive up to 2 decimal point and less than 1 billion.
> - `YEAR_MONTH` is in the format `YYYY-MM`, which is the year and month for the budget. If not specified, the current year and month will be used.

NOTE	Budget cannot be unset after setting it.

Examples:

- `setbudget -a 1000`
  Sets the budget of $1000 for this month.
- `setbudget -a 1000 -ym 2020-03`
  Sets the budget of $1000 for the month of March in 2020.

4.7. Locating both repeats and normal expenditures by keyword: `find`

Find expenditures (both single and repeated) which contain the keyword(s).
Format: `find [KEYWORD…]`

- The search is case insensitive. e.g `chickens` will match `Chickens`
- The order of the keywords does not matter. e.g. `Chicken Rice` will match `Rice Chicken`
- Substrings will be matched e.g. `Chicken` will match `Chickens`
- Expenditures matching at least one keyword will be returned (i.e. `OR` search). e.g. `Chicken Rice` will return `Fried Chicken`, `Steam Chicken`

TIP Use `exp list` to exit from the search results.

Examples:

- `find rice`
  Returns `Chicken rice` and `Veg Rice`.
- `find Spicy Chicken Rice`
  Returns any expenditures or repeats having names `Spicy`, `Chicken`, or `Rice`.

4.8. Report commands

This section will talk about how you can use report feature. Before we begin with the details of the command, here are some tips:

1. Before beginning this section, you can refer to Report to look at how the reports will look like and how to read it.
2. If there is a huge difference in expenditure amounts, using a Bar chart will be better as overlaps might occur for Pie chart.
3. For best usage, keep start date and end date to be **within 12 months** for reports generated by months. Similarly, keep number of tags to be **within 12** for reports generated by tags. It is possible that not all months or tags will be displayed, if start and end date exceeds 12 months or tags exceed 12.
4. Do read the NOTE section in the command, if any.

4.8.1. Viewing expenditures report: `report view`

Generates report on expenditure spending in the given period.

Format: `report view -sd START_DATE -ed END_DATE -g GRAPH_TYPE -o ORGANISATION`

- Views the report of a graph type populated with expenditure data from start date to end date.
- `START_DATE` is the date from which the report will start generating from.
- `END_DATE` is the date from which the report will stop generating.
- `GRAPH_TYPE` is the type of the graph you want to generate. Possible values are:
  - `bar`: bar graph
  - `pie`: pie chart
- `ORGANISATION` is how the expenditures will be grouped when generating the graph. Possible values are:
  - `tag`: organised by tags
  - `month`: organised by months

Examples:

- `report view -sd 2020-03-01 -ed 2020-03-31 -g pie -o tag`
- `report view -sd 2020-03-01 -ed 2020-03-31 -g bar -o month`

4.8.2. Exporting report: `report export`

Exports report on expenditure spending in the given period.

Format: `report export -sd START_DATE -ed END_DATE -g GRAPH_TYPE -o ORGANISATION -f FILE_NAME`

- Exports the report of a graph type populated with expenditure data from start date to end date into Report folder with file's name being FILE_NAME.
- `START_DATE` is the date from which the report will start generating from.
- `END_DATE` is the date from which the report will stop generating.
- `GRAPH_TYPE` is the type of the graph you want to generate. Possible values are:
  - `bar`: bar graph
  - `pie`: pie chart
- `ORGANISATION` is how the expenditures will be grouped when generating the graph. Possible values are:
  - `tag`: organised by tags
  - `month`: organised by months
- `FILE_NAME` will be the name of the file.

NOTE	The exported file will be located at the same directory of the application, under the `Report` folder. The file can be identified by `FILE_NAME.png` whereby `FILE_NAME` is specified by your input in the command.

Examples:

- `report export -sd 2020-03-01 -ed 2020-03-31 -g pie -o tag -f hello`
- `report export -sd 2020-03-01 -ed 2020-03-31 -g bar -o month -f report`

4.8.3. Printing report: `report print`

Prints report on expenditure spending in the given period.

Format: `report print -sd START_DATE -ed END_DATE -g GRAPH_TYPE -o ORGANISATION`

- Prints the report of a graph type populated with expenditure data from start date to an end date.
- Refer to Section 4.8.1, "Viewing expenditures report: `report view`" for the details on the parameters.

NOTE	For best usage, avoid sending multiple print jobs in short period of time. This prevents the print jobs from getting lost.

Examples:

- `report print -sd 2020-03-01 -ed 2020-03-31 -g pie -o month`
- `report print -sd 2020-03-01 -ed 2020-03-31 -g bar -o tag`

4.9. Exiting the program: `exit`

Exits the program.
Format: `exit`

5. Report Window Commands

These are the commands to be executed in the report window. Before we begin with the details of the command, here are some tips:

1. Before beginning this section, you can refer to Report to look at how the reports will look like and how to read it.
2. If there is huge difference in expenditure amount, using a Bar chart will be better as overlaps might occur for Pie chart.
3. For best usage, keep start date and end date to be **within 12 months** for reports generated by months. Similarly, keep number of tags to be **within 12** for reports generated by tags. It is

possible that not all months or tags will be displayed, if start and end date exceeds 12 months or tags exceed 12.

4. Do read the NOTE section in the command, if any.

5.1. Viewing expenditures report

NOTE The result of this command is equivalent to Section 4.8.1, "Viewing expenditures report: `report view`".

Generates report on expenditure spending in the given period.
Format: `view START_DATE END_DATE GRAPH_TYPE ORGANISATION`

- Views the report of a graph type populated with data from expenditures from a start date to an end date.
- `START_DATE` is the date from which the report will start generating from.
- `END_DATE` is the date from which the report will stop generating.
- `GRAPH_TYPE` is the type of the graph you want to generate. Possible values are:
  - `bar`: bar graph
  - `pie`: pie chart
- `ORGANISATION` is how the expenditures will be grouped when generating the graph. Possible values are:
  - `tag`: organised by tags
  - `month`: organised by months

Examples:

- `view 2020-03-01 2020-03-31 pie tag`
- `view 2020-03-01 2020-03-31 bar month`

5.2. Exporting report: `export`

Exports the current report shown in the report window.
Format: `export FILE_NAME`

- Exports the report to Report folder with file's name specified by user
- `FILE_NAME` will be the name of the file.

NOTE The exported file will be located at the same directory of the application, under the `Report` folder. The file can be identified by `FILE_NAME.png` whereby `FILE_NAME` is specified by your input in the command.

5.3. Printing report: `print`

Prints the current report shown on the report window.
Format: `print`

> **NOTE** For best usage, avoid sending multiple print jobs in short period of time. This prevents the print jobs from getting lost.

5.4. Help: `help`

Shows the user the commands they can enter.
Format" `help`

5.5. Exiting the report window: `exit`

Exits the report window.
Format: `exit`

6. FAQ

Q: How do I transfer my data to another Computer?
A: Install the app in the other computer and overwrite the empty data file it creates with the file that contains the data of your previous $AVE IT folder.

7. Command Summary

7.1. Prefix Notation

Prefix	Meaning
`-i`	information
`-a`	amount
`-d`	date
`-t`	tag
`-sd`	start date
`-ed`	end date
`-ym`	year month
`-p`	period
`-g`	graph type
`-f`	file name
`-o`	organised by

7.2. General Operations

- **help** : `help`
- **go** : `go DATE`
  e.g `go 2020-04-01`
- **find** : `find [KEYWORD…]`
  e.g. `find chicken rice`
- **setbudget** : `exp setbudget -a AMOUNT [-ym YEAR_MONTH]`
  e.g `setbudget -a 1000 -ym 2020-04`
- **exit** : `exit`

7.3. Account Operations

- **acc add** : `acc add ACCOUNT`
  e.g. `acc add Personal`
- **acc list** : `acc list`
- **acc rename** : `acc rename OLD_NAME NEW_NAME`
  e.g. `acc rename Personal non-personal`
- **acc checkout** : `acc checkout ACCOUNT`
  e.g. `acc checkout Personal`
- **acc clear** : `acc clear`
- **acc delete** : `acc delete ACCOUNT`
  e.g. `acc delete Personal`

7.4. Expenditure Operations

- **exp add** : `exp add -i INFO -a AMOUNT [-t TAG] [-d DATE]`
  e.g `exp add -i chicken rice -a 3.50 -t meal -d 2020-04-01`
- **exp edit** : `exp edit INDEX [-i INFO] [-a AMOUNT] [-t TAG] [-d DATE]`
  e.g `exp edit 1 -i duck rice -a 4.50 -d 2020-04-02`
- **exp delete** : `exp delete INDEX`
  e.g `exp delete 1`
- **exp list** : `exp list`

7.5. Repeat Operations

- **repeat add** : `repeat add -i INFO -a AMOUNT -sd START_DATE -ed END_DATE -p PERIOD [-t TAG]`
  e.g `repeat add -i bus fare -a 1.50 -sd 2020-03-01 -ed 2020-04-01 -p daily -t transport`
- **repeat edit** : `repeat edit INDEX [-i INFO] [-a AMOUNT] [-sd START_DATE] [-ed END_DATE] [-p PERIOD] [-t TAG]`
  e.g `repeat edit 2 -a 1.20 -ed 2020-04-02`

- **repeat delete** : `repeat delete INDEX`
  e.g `repeat delete INDEX`

7.6. Report (Main Window) Operations

- **report view** : `report view -sd START_DATE -ed END_DATE -g GRAPH_TYPE -o ORGANISATION`
  e.g. `report view -sd 2020-03-01 -ed 2020-03-31 -g pie -o month`
- **report export** : `report export -sd START_DATE -ed END_DATE -g GRAPH_TYPE -o ORGANISATION -f FILE_NAME`
  e.g. `report export -sd 2020-03-01 -ed 2020-03-31 -g pie -o month -f hello`
- **report print** : `report print -sd START_DATE -ed END_DATE -g GRAPH_TYPE -o ORGANISATION`
  e.g. `report print -sd 2020-03-01 -ed 2020-03-31 -g bar -o tag`

7.7. Report (Report Window) Operations

- **report view *equivalence*** : `view START_DATE END_DATE GRAPH_TYPE ORGANISATION`
  e.g. `view 2020-03-01 2020-03-31 bar tag`
- **report export *equivalence*** : `export FILE_NAME`
  e.g. `export hello`
- **report print *equivalence*** : `print`
  e.g. `print`
- **exit** : `exit`
  e.g. `exit`