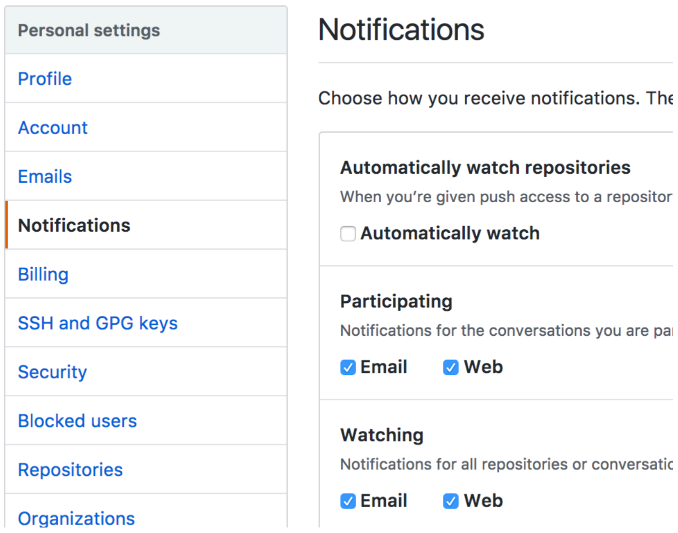
Practical 5 - Dictionaries, Code Reviews with PRs

Did you finish last week's work? Including the practice section? If not, make sure to complete it during the week. If you do not understand anything, bring those questions to your tutor the following week.

Your reason for doing this subject/degree probably has something to do with getting a job in the IT industry. Our reason for teaching you things like we are today is the same – to prepare you for work in the industry.
A common and important part of development jobs in the IT industry is doing **code reviews**, where a peer evaluates and comments on another's work with the goal of improving the final result (and mutual learning).
So, to help you towards being a better programmer and being more familiar with industry practices, today you will work on some programs, then review another student's work.

*You need to complete the tasks and do a **Pull Request** that mentions another student to get full marks.*
This means you need a partner so you can review each other's code. **Organise someone right now and swap GitHub usernames** (external students do this via Slack). It's also OK to review multiple people's code or to have a cycle of people reviewing the next person's... it doesn't have to be a direct swap.

First, **edit your notifications settings**:
Go to GitHub, login, then click on your account link (top-right) and choose Settings, then Notifications.
Click to select the "Web" and "Email" options so that you will receive notifications when you're mentioned.



Notifications

Choose how you receive notifications. The

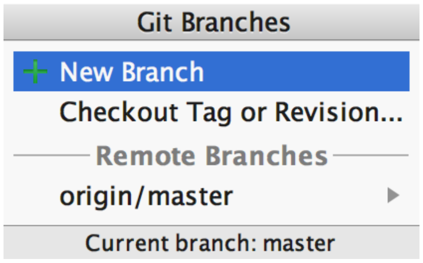
Automatically watch repositories
When you're given push access to a repositor
☐ Automatically watch

Participating
Notifications for the conversations you are pa
☑ Email ☑ Web

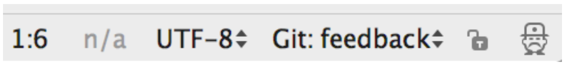
Watching
Notifications for all repositories or conversatic
☑ Email ☑ Web

Code reviews are often done via **pull requests** on GitHub **branches**.
A branch is another copy or version of an entire repository. Branches are useful for things like working on new features or bug fixes, or in our case getting feedback.

1. Open your PyCharm Practicals project. (If it's not already up to date on GitHub, then commit and push your current work now.)
2. Create a branch called "feedback" by: **VCS > Git > Branches** then click **+ New Branch**



It may look like nothing happened, but your local repository now has a new branch, which is currently exactly the same as the 'master' branch, and it is now the one that is "checked out". See in the footer:



So now the work you do and commit will be in the feedback branch, not the master. We will then do a pull request from feedback to master, which is basically a request for someone to **merge** the changes in feedback into the master branch... which gives us the opportunity to provide comments via GitHub.

So, you're on the feedback branch? Great! Let's write some code using **dictionaries**...

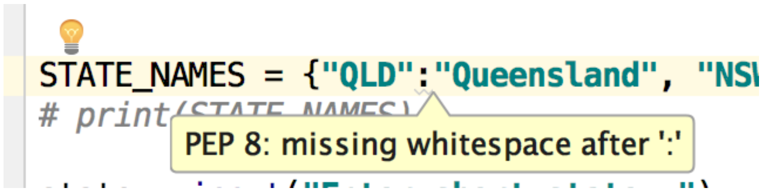
Walkthrough

Copy the code from: https://github.com/CP1404/Practicals/blob/master/prac_05/state_names.py
This is a program that uses a ‘constant’ (name is ALL_CAPS) dictionary to store the Australian state abbreviations and names - e.g. QLD is Queensland. It asks the user for their 'short' state and prints the full state name by looking it up in the dictionary.

Things to do:

1. Run the program to see how it works.

2. Right now the code formatting of this dictionary is incorrect and inconsistent. Dictionary literals should be formatted with no space before, and one space after, the colon, like `{"A": 1, "B": 2}`
   Thankfully, PyCharm knows this and can fix it for you! If you move your mouse over the grey line near the first colon, PyCharm pops up the problem... Click on it to see the action icon (light bulb) to the left... Click on that to see the options... Choose reformat file... Problem solved.
   You can also choose **Code > Reformat Code** or use the keyboard shortcut any time. It does the whole file or whatever's selected. No more excuses for dodgy formatting!
   This is a great way to learn what the "PEP 8" standards are for Python code formatting style.

   ```
   STATE_NAMES = {"QLD":"Queensland", "NS
   # print(
   ```

   PEP 8: missing whitespace after ':'

3. Currently the program requires you to enter the states in capitals. Change the program so lowercase inputs also work to show the state names. (There are two places to add a string method.)

4. Do this next part on paper first (then in PyCharm: Write a loop that prints all of the states and names neatly lined up with string formatting, like:

```
NSW  is New South Wales
QLD  is Queensland
NT   is Northern Territory
```

Intermediate Exercises

Based on the state name example program above, create a program that allows you to look up hexadecimal colour codes like those at http://www.color-hex.com/color-names.html
Use a constant dictionary of about 10 names and write a program that allows a user to enter a name and get the code, e.g. entering **AliceBlue** should show **#f0f8ff**.

Do-from-scratch Exercise

Write a program to count the occurrences of words in a string. The program should ask the user for a string, then print the counts of how many of each word are in the file.
The output should look like this:

```
Text: this is a collection of words of nice words this is a fun thing it is
a : 2
collection : 1
fun : 1
is : 3
it : 1
nice : 1
of : 2
thing : 1
this : 2
words : 2
```

Hints: use a dictionary where the keys are the words and the values are the counts; when you find a word, check if it’s in the dictionary...

- Notice that the sample output is sorted. As a refinement step, **after** you have the program working, **make your program do this sorting**.

- As a further refinement, **align the numbers so they are in one nice column**. You will need to find the longest word in the list first, and then you need to know how to use the str.format method to take a variable width. The former is up to you, but the latter can be done with another {} placeholder, like:

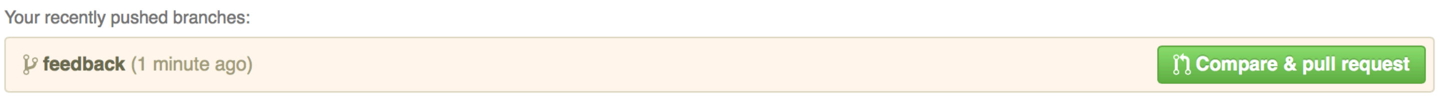
  ```
  print("{:{}} = {}".format(x, y, z))
  ```

  This formats the first placeholder value, x, with a width of y then prints a literal = then the value of z. Nice.

Code Reviews with Pull Requests

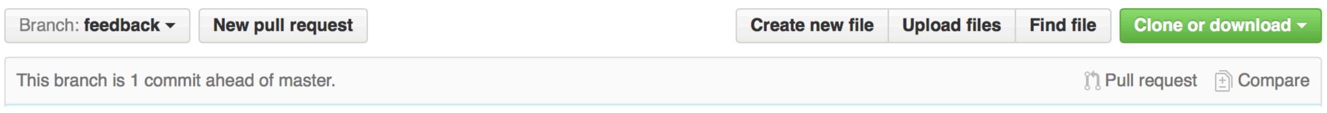
This is a process based on how code reviews and pull requests (PRs) happen in the IT industry but simplified to suit our teaching environment.

1. Commit your changes making sure to add any new files that you created today.
   You have already created and switched to the 'feedback' branch so your commits will go only to this branch.
2. Open the repository in a web browser and you should see a notice like:


Your recently pushed branches:

feedback (1 minute ago)

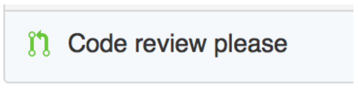
3. Click the green button to make a pull request from feedback to master.
   If that notice doesn't appear, you can switch to the feedback branch and click Pull Request:


This branch is 1 commit ahead of master.

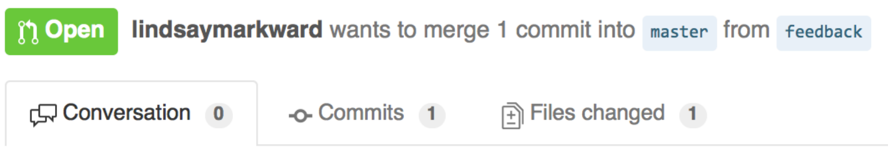
   Add a title like "Code review request" and some detail like "check formatting, naming and logic" (or anything else you want checked).
   In this description, mention the reviewer with their GitHub username and the @ symbol – e.g. @jondo – so they will be notified (depending on their GitHub preferences)

OK, now if you are the first to do this, you're finished for now... move on to the next section and come back here when you receive a mention to do a review for someone else...
If you've already got a PR to review, then carry on with the next steps:

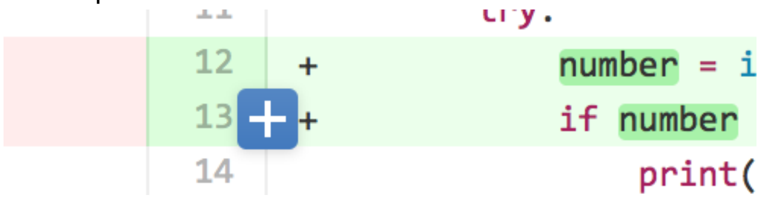
4. (Note that you might see things a bit different, but the process is the same.)
   On the GitHub website, click on the notifications icon at the top: which will have a dot on it if you've received your code review request.
   Open the PR you have been mentioned in by clicking the notification link there, e.g.:


Code review please

   Read the request (see if there's anything specific to review), then click on Commits to see the commits:


lindsaymarkward wants to merge 1 commit into `master` from `feedback`

   Then click on the commit to see the code in "diff" view.
5. Read through the code on GitHub and add line comments. Hover your mouse over the lines and look for the plus icon to add a comment.

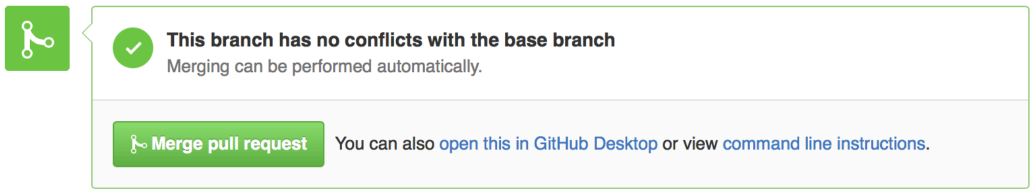


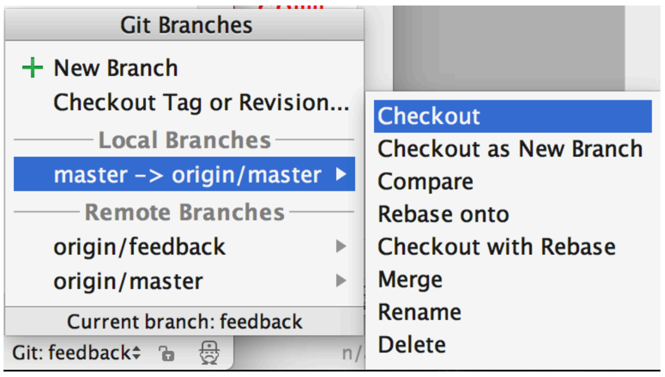
```
        try:
+           number = i
+           if number
                print(
```

Your job is to look for anything that could be improved including incorrect, inconsistent or non-ideal naming, formatting, logic... anything relevant. Add clear explanations, suggestions or questions. **Important:** this is not a trivial exercise. Don't just write "all good mate :)", but take your time and add thoughtful comments that help you and the requester to learn and improve.

(When you've finished, there's nothing more to do as part of this review process. In a collaborative environment where you and the reviewer have push access to the repository, there would be more, including the option to merge and close the pull request.)

OK, so at some point you will receive the comments from the reviewer and you can respond to them by making changes in your own code in PyCharm and replying to the comments on GitHub. Then commit your work back to GitHub, still in the feedback branch.

6. Ideally, the reviewer would re-check this new work after the updates and make more comments... then the author does more work if need be... reviewer adds more comments... until all good (the reviewer decides when it's finished) ... then the reviewer would close the pull request.
   In our simplified version of this process, you can just **Merge the pull request** now (add a comment if you want):
   
   This branch has no conflicts with the base branch
   Merging can be performed automatically.

   You can also open this in GitHub Desktop or view command line instructions.
   GitHub will tell you that the feedback branch can be deleted. You're welcome to delete it.
7. Now the master branch has been updated on GitHub, but not locally.
   You should switch back to the master branch locally by clicking in the footer where it shows the branch:
   

8. Pull your changes from master (remote) to update your local repo: **VCS > Git > Pull** then click **Pull**.

All done!
What did you learn from this?
To read more about Pull Requests: https://help.github.com/articles/using-pull-requests/

Not all done :) Now on to more practice programming…

Practice & Extension Work

1. Convert parallel lists into a dictionary…
   Recall that it's possible to represent information in the form of parallel lists where the indices determine how the information is related across lists. For example:
   ```
         names = ["Jack", "Jill", "Harry"]
         dobs =  [(12, 4, 1999), (1, 1, 2000), (27, 3, 1982)]
   ```
   This means Jack was born on 12/4/1999, Jill was born on 1/1/2000, and Harry was born on 27/3/1982. Write a program using a dictionary instead of the above parallel lists that allows the user to enter the date-of-birth details for 5 people, and have it display their individual ages.
   Hint: you can **split()** a string like "12/4/1999", as we did in the lecture last week.

2. Write a function that takes two parallel lists as input parameters and returns a dictionary where keys are from the first list and the values are from the second. Use the above example as a test case.

3. In practical 1 you should have created an electricity bill estimator using constant values for the tariff amounts like:
   ```
   TARIFF_11 = 0.244618
   TARIFF_31 = 0.136928
   ```

   ```
   Electricity Bill Estimator 2.0

   Which tariff? 11 or 31: 11
   Enter daily use in kWh: 13.4
   Enter number of billing days: 90

   Estimated bill: $295.01
   ```

   Now create a version of the above electricity program that uses a **dictionary** to store the tariffs and the corresponding cost.
   In the prompt, list all of the tariffs (all of the dictionary keys) and make sure a valid one is selected.
   Use the appropriate cost from the dictionary to calculate the bill total.
   You will need to change how you present the "Which tariff" prompt, since these values come from the dictionary.

   To show the benefit of this, add three more tariffs (make them up).
   You should find that this is a very simple step for you, and your program can handle it without any extra coding.