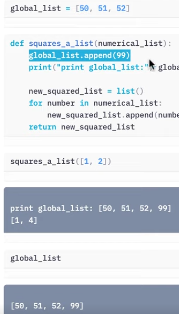
Python Notes Modules 4-6

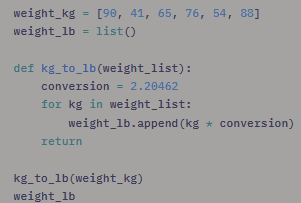
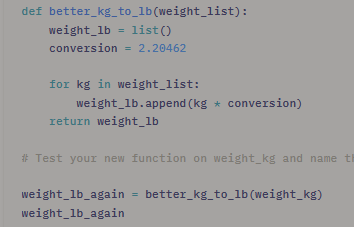
DRY Revisited

* Don’t repeat yourself
* Scoping
  + Local variable—created inside function and only exist inside function , also know as the “scope” of the function
  + Global variables—recognised both inside and outside a function
  + If a variable inside a function has the same name as a variable outside a function, the inside variable gets printed when the function gets executed
    - It doesn’t overwrite the outside variable of the same name; outside its function, the outside variable gets printed
  + Can even append global variables from inside a function

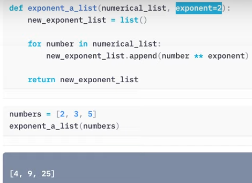
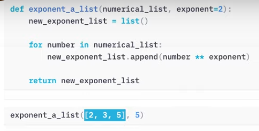
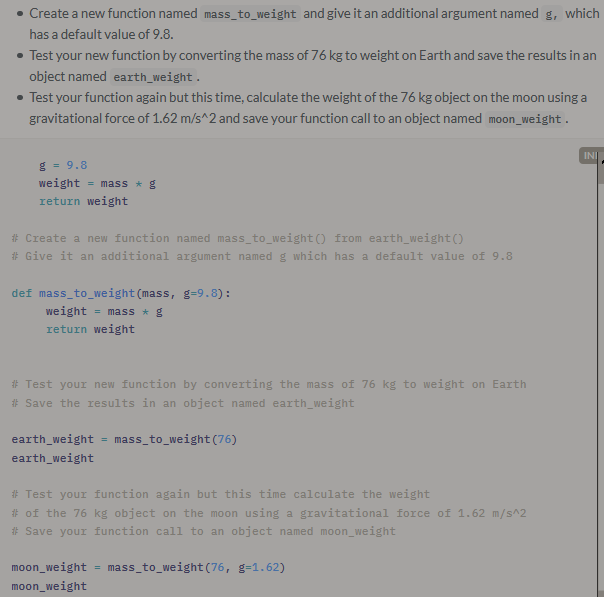


* .drop() – creates a new df with the columns/rows dropped
* Inplace – decides whether or not df calling function is going to be modified or not
  + Affects original df, making it the same as the copy
* Side effects – does not create a new file – no ‘=

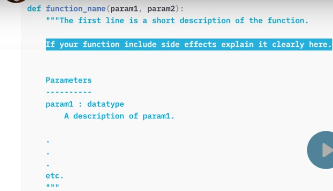
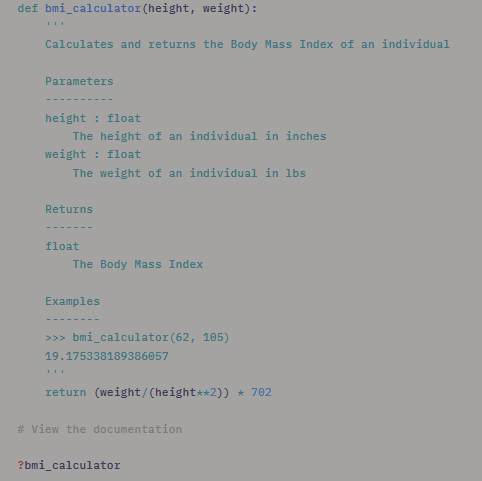
‘ sign needed

* 
  + Weight\_lb , which is outside of the function kg\_to\_lb, has been altered by the appendment within that function
* To avoid this, try something like this instead:
  + 

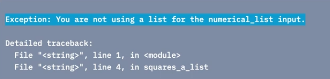
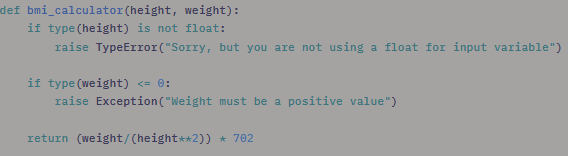
Default Arguments

* Exponent – takes 2 arugments (numerical list and exponent)
  + Can be used for any exponent we want
    - E.g. exponent = 2
      * When applied to a list, it squares the list
    - E.g. 
  + From then on, you can just write exponent and it will square by default
  + Arguments that come after a default argument must also have default values
    - Can change value to a different exponent as well: 
* Argument Ordering
  + 
  + 2, 3, 5 belongs to numerical list while and 5 belongs to exponent
* E.g.
  + 

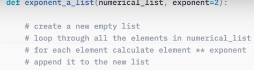
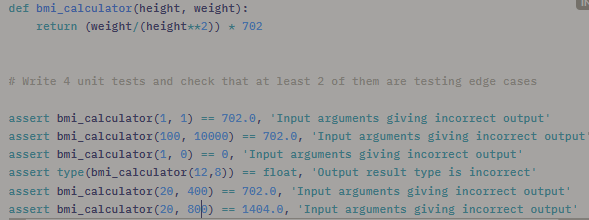
Function Docstrings

* String that comes after def line and documents function’s purpose and usage
  + Helps explain to others what it does
  + NumPy format of docstrings contain 4 main sections
    - 1. A brief description of the function
    - 2. Explaining the input parameters
    - 3. What the function Returns
    - 4. Examples
  + Has no impact on the code
* Side effects
  + Clearly state them in your documentation
  + 
  + E.g. 

Defensive Programming Using Exceptions

* Write a code in such a way that if errors occur, they do so in a ‘graceful’ manner
* Exceptions – used in defensive programming – disrupts the normal flow of instructions. When Python encounters code that it cannot execute, it will throw an exception
  + Useful when working with others who want to modify your work
  + Better than receiving error message since it explains exactly what’s wrong
* E.g. 
  + 
* Why raise exceptions
  + Check datatypes – makes sure correct datatypes are being employed
* more specific exceptions – TypeError –wrong datatype
* exception documentation
  + set the expectations for the users as much as possible
  + 
    - NB: Raises goes between Returns and Examples
* E.g. 

Unit Tests

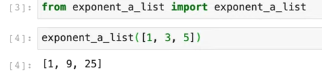
* Assert statements
  + Checks that something is true
  + 
  + Useful for testing expectations of functions
    - Helps one specify what needs to be done
    - Helps firm up what you want to do
  + What to test?
  + E.g. 
* Systematic approach
  + Used to design our function using a general set of steps to follow when writing programs
    - 1. write a function stub (first line of input)
    - 2. Write tests to satisfy the design specifications
    - 3. Outline the program with pseudo-code
      * E.g. 
    - 4. Write the code and test frequently
    - 5. Write documentation
  + Advanced example: 
* Good Function Design Choices
  + What makes a function useful?
  + Avoid hard coding
    - Process of embedding values directly into your code w/o saving them in objects
    - Specific values that are subject to change, amking the function useless should the value change (e.g. exchange rate)
  + Less is more—don’t’ have your functions do too many things at the same time
    - Don’t lock it into a specific dataframe
  + Return a single object
    - While functions can return multiple objects, but this can get confusing
  + Keep global variables in their global environment

**Module 7**

Importing Python Libraries

* Specify library name, then use function
* From – imports single function
  + From pandas import read\_csv
  + Lets you import function directly

Working with Other Files

* How to you use a function that exists in another file?
  + Instead of copying and pasting the function every time you wish to use it, you can create a python script and put a function inside that and then import it
  + Step1—create text file
    - Make it a py file
    - Copy and paste function into new file
    - In destination file, write where function is
      * From exponent\_a\_list (py file source) import exponent\_a\_llist (name of function contained in py file)
      * E.g. 

Testing Your Own Functions with Pytest

* Move your tests to different files as well
  + Put them in your own file
  + Create new text file
    - Rename and change to .py file
    - Copy and paste testing codes
      * Wrap them inside a function
        + E.g. def test\_exponent\_a\_list

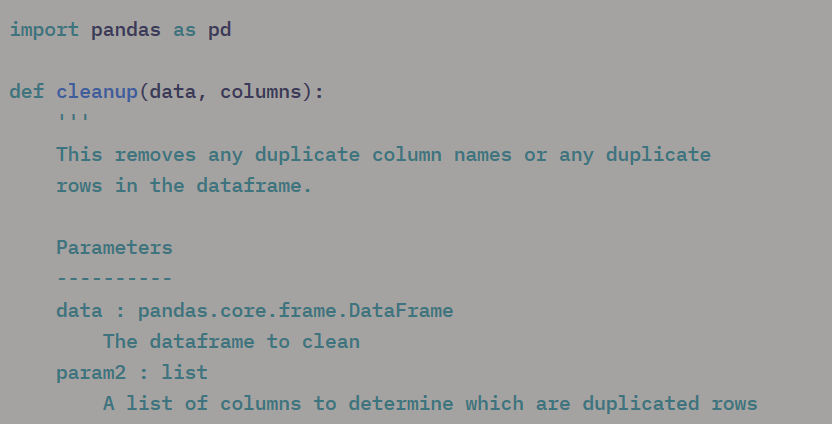
Add import statement to explain where function is

From exponent\_a\_list import exponent\_a\_list

Save

* Open terminal, run: pytest
  + Runs functions that start with test as well
    - Can be used to verify whether tests have passed

Automatic Style Formatting

* Make your code easily understood by other people
  + Code is read much more often than it is written
  + PEP8 style guide
    - Indent using 4 spaces
    - Having whitespace around operator e.g. x = 1 not x=1
    - Avoiding extra whitespace (f(1) not f (1))
    - Single and double quotes for strings, but only using triple double quotes, not triple single quotes
    - Varaiable and function names using underscores\_between\_words
  + Flake8 – ‘grammar check’ for code
    - Will make suggestions, but not make changes
  + Black—tool that will follow your code for you
    - Uses Pep8 formatting for the most part
  + use terminal to use flake8 –
    - e.g. flake8 <file name>
      * to check other files, use flake8-nb <file name>
      * to do it in Jupyter Notebook:
        + !flake8 <file name>
        + Or use black -- !black <file name>
  + Use terminal with black
    - Black <file name>
  + Format directly using black
    - Right click and format cell
      * Formats cell immediately
  + Example:
  + 
  + 