CSCA08 Fall 2017

Week 6 - Files & Mutability

Brian Harrington & Marzieh Ahmadzadeh

University of Toronto Scarborough

October 16 - 20, 2017

UNIVERSITY OF TORONTO SCARBOROUGH

FILES - OPENING

- Opening a file:
- open(`filename`,`mode`)
  - `(str,str) -> io.TextIOWrapper`
  - opens the file `filename`, in the same directory as the .py file
  - returns a `file handle`
  - `mode` can take several values:
    - `r`: open the file for reading
    - `w`: open the file for writing (erasing whatever was in it... careful with this one!)
    - `a`: open the file for writing, but appending new information to the end of the file

Files - Closing

- Closing a file:
- filehandle.close()
  - Note that this is a method, not a function
  - closes the file (good form, often forgotten)

Reading a File

- `filehandle.readline()` - read 1 line from the file
- `filehandle.read()` -read the whole file into a single string
- `filehandle.readlines()` - read the whole file into a list, with each element being one line of text
- `filehandle.readlines(n)` - read the next `n` bytes of a file, rounded up to the end of a line
  - We won't be using this one much

Reading a File: Different Methods

- Examples

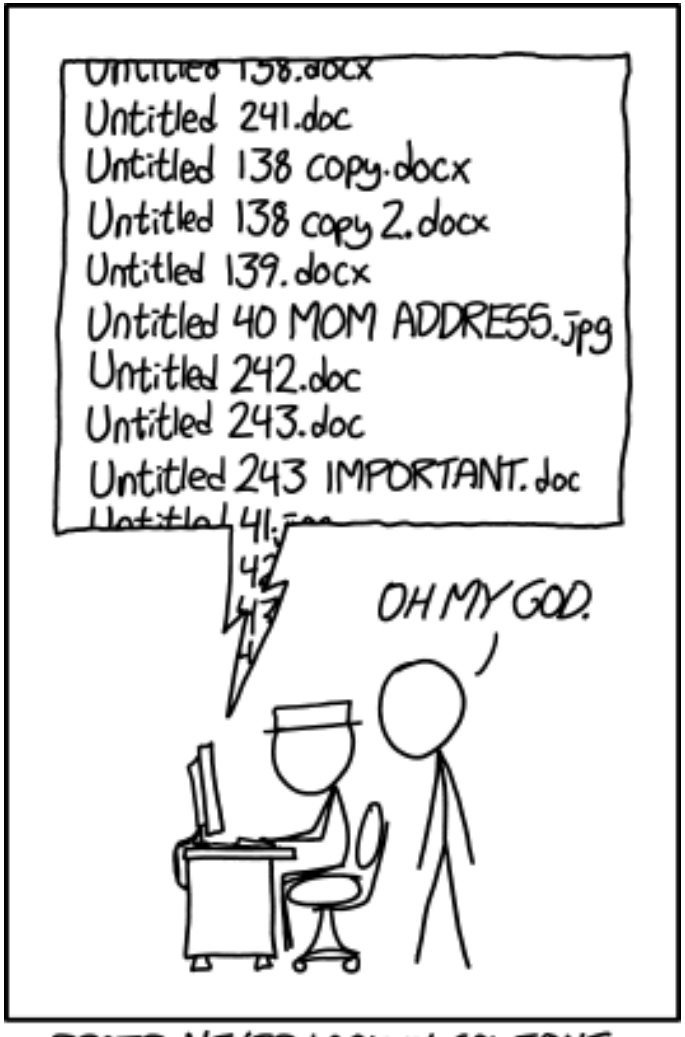
Writing to a File

- `filehandle.write()`
- As easy as printing
- Except you have to add your own newline characters
- And it only takes strings (anything else, just convert yourself)
- Don't forget to close your file, or it may not write

LET'S HAVE SOME FUN

LET'S HAVE SOME FUN

Break



PROTIP: NEVER LOOK IN SOMEONE ELSE'S DOCUMENTS FOLDER.

The big reveal

- Recall from week 2: functions cannot change global variables
- This must be correct... we tried it
- And surely Brian wouldn't lie to us...
- Would he?

Mutability

- `int`s, `string`s, `boolean`s, ets are **immutable**.
  - We can't change them, we can only make new ones
  - `counter += 1` actually creates a new value for counter to point to every time
- Lists are **mutable**, we can change their values without creating a whole new list
- Let's see an example

Aliasing

- Since variables are really just references to memory address, `x = y` really means `point x to the same spot in memory that y is pointing to`
- This is called `aliasing`, making y an `alias` of x
- It means that if `x` changes, `y` will change as well

Cloning

- What if we don't want to create an alias?
- We can create a *clone* instead
- `my_new_list = my_old_list[:]`
  - logically: create a new list copying items from start to end of the old list
  - makes a new copy of the values inside the list
  - won't work for lists of lists