hw1pr1: *Files!*

- walking through folders and files (500 files, even!)
- analyzing, counting, inquiring, and *discovering* as we go!

In [1]:
```
# Where are we?
%pwd
```

Out[1]:
```
'/Users/MyDocuments/MyPython/College Courses/Harvey Mudd CS35/Sci50 2024 Spring/Homework/W
eek1/week1_spr24/hw1pr1'
```

By the way, when I run the above cell on my desktop machine (Mac), the response is

`'/Users/zacharydodds/Desktop/sci50/week1_spr24_v1/problem1'`

on a Windows laptop, the path uses a different separator. Mine is

`'c:\\Users\\dodds\\OneDrive\\Desktop\\sci50'`

Your results will almost certainly differ (if they don't, I'm interested! :)

In [2]:
```
# what's here?
%ls
```

```
file42.txt*              phonebook 2013/          phonebook 2019/
hw1pr1.ipynb             phonebook 2014/          phonebook 2020/
intro_first/             phonebook 2015/          phonebook 2021/
intro_first_ss_small.png phonebook 2016/          phonebook 2022/
intro_second/            phonebook 2017/          phonebook 2023/
phonebook 2012/          phonebook 2018/          phonebook 2024/
```

In [3]:
```
# to move around:  cd stands for "change directory" (a directory is a folder)
#    %cd intro_first    would move into the intro_first folder
#    %cd .. moves "up" to the containing directory
#    %cd .  doesn't move at all:  .  represents the current directory

# For now, let's not move anywhere
%cd .
```

```
/Users/MyDocuments/MyPython/College Courses/Harvey Mudd CS35/Sci50 2024 Spring/Homework/We
ek1/week1_spr24/hw1pr1
```

In 2024, my Mac setup succeeds with a `UserWarning` that tells me to install the `pickleshare` library.

I refuse on snack-sharing principle. One can only share freely, not under admonishment/warning!

In [4]:
```
# we will use a few file-handling "system" libraries.
# These are built-in to python, so nothing to install - just to import:
import os
import os.path
```

On first glance, it seems we can't open the file `nottrue.ipynb` ...

In [5]:
```
#### But that's not true!  Let's look at the contents of nottrue.ipynb
#    There are different commands for Mac and Windows. (Linux is like Mac)
#    Uncomment the one matching your system:

print("+++ Contents of the file nottrue.ipynb: +++\n")

# Mac:    !cat  <filepath>  using forward slashes
#
# !cat ./intro_first/nottrue.ipynb
# Windows:  type <filepath>  using backslashes
#
# !type .\\intro_first\\nottrue.ipynb
```

```
+++ Contents of the file nottrue.ipynb: +++
```

Indeed, we can use the command-line `cat` or `type` one file at a time...

But, what if we have to walk *500 files* ?! (Alas, this joke stays around this whole week!)

- Then, we need a function - and script - to access its contents.
- We used it last week, let's use it again here:

In [6]:
```
#
# function to return the contents of a file (as one-big-string)
#

def GET_STRING_FROM_FILE(filename_as_string):
    """ return all of the contents from the file, filename
        will error if the file is not present, then return the empty string ''
    """
    try:
        # the encoding below is a common default, but not universal...
        file_object = open(filename_as_string, "r", encoding='utf-8')    # open! (Other en
        file_data = file_object.read()                                   # and get all its
        file_object.close()                                              # close the file
        #print(DATA)                                                     # if we want to s
        return file_data                                                 # definitely want
    except FileNotFoundError:                              # it wasn't there
        print(f"file not found: {filename_as_string}")    # print error
        return ''                                          # return empty string ''
    # except UnicodeDecodeError:
    #     print(f"decoding error: {filename_as_string}")    # encoding/decoding error
    #     return ''                                         # return empty string ''


full_file_path = "./intro_first/nottrue.ipynb"
file_contents = GET_STRING_FROM_FILE(full_file_path)      # reminder: file_contents = file

# Let's print only some of this potentially large string, adapting as needed:
print("file_contents:\n\n", file_contents[0:42])          # let's then increase the 42...
```

```
file_contents:

 Hi, everyone in Sci50!

As you're seeing,
```

Notice that, in Python, the Mac/forwardslash/style paths work, *even on Windows*

In [7]:
```
####  Let's try one of the other files!  (or a non-existent file!)

full_file_path = "./intro_first/cs/file35.txt"    # how about the others?!
file_contents = GET_STRING_FROM_FILE(full_file_path)
print("file_contents:\n\n", file_contents[0:42])
```

```
file_contents:

 CS for Insight
```

But, we have 500 files...

Let's write _steppingstone_ functions to make sense of our 500 files...

Let's start by reminding ourselves we can write a function that returns.

We'll call this Version 0:

In [8]:
```
#
# Steppingstone, Version 0: does Python work?
#

import os
import os.path

def file_walker(path):
    """ starting from the input, named path

        this function "walks" the whole path, including subfolders
        and then... explores any questions we might want :)

        call, for example, with    file_walker("./intro_first")
    """
    return 42  # just to check that it's working (v0)

#
# when discovering, keep your data close (and your functions closer!)
#
if True:
    """ overall script that runs examples """
    print(f"[[ Start! ]]\n")     # sign on

    path = "./intro_first"       # Remember: . means the current directory
    result = file_walker(path)   # Run!

    print(f"result = {result}")  # Yay for f-strings!

    print("\n[[ Fin. ]]")        # sign off
```

```
[[ Start! ]]

result = 42

[[ Fin. ]]
```

Introducing `os.walk`

The function `os.walk(path)` will walk *any number* of files...

Before we write a function, let's try `os.walk` immediately:

In [9]:
```
# os.walk returns the structure of a folder (directory)

# Here, we "walk" the intro_examples subfolder:
all_files = os.walk("./intro_first")

all_files     # oops! it's a "generator object"
```

Out[9]:
```
<generator object walk at 0x7fa2a1ec06d0>
```

In [10]:
```
import os
L = list( os.walk( "./intro_first" ) )
print(f"{len(L)}")
print(f"{L}")
```

```
5
[('./intro_first', ['cs', 'sci'], ['.DS_Store', 'nottrue.ipynb']), ('./intro_first/cs',
[], ['.DS_Store', 'file35.txt', 'file181y.txt']), ('./intro_first/sci', ['50', '10'], ['.D
S_Store']), ('./intro_first/sci/50', [], ['IDE.txt']), ('./intro_first/sci/10', [], ['IDE.
txt'])]
```

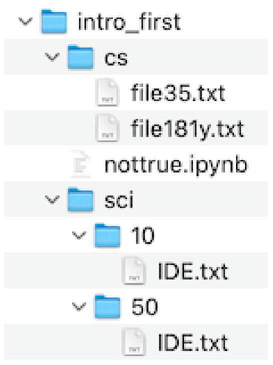
Here's a line-wrapped version of the list `L`

- Below it, is a picture of the folder-and-file structure!
- Our goal: mind-mapping the two representations!!

`L = [('./intro_first', ['cs', 'sci'], ['.DS_Store', 'nottrue.ipynb']), ('./intro_first/cs', [], ['.DS_Store', 'file35.txt', 'file181y.txt']), ('./intro_first/sci', ['50', '10'], ['.DS_Store']), ('./intro_first/sci/50', [], ['IDE.txt']), ('./intro_first/sci/10', [], ['IDE.txt'])]`

In [11]:
```python
from IPython import display
display.Image("./intro_first_ss_small.png")   # local image
```

Out[11]:



```
intro_first
  cs
    file35.txt
    file181y.txt
  nottrue.ipynb
  sci
    10
      IDE.txt
    50
      IDE.txt
```

See if you can match the *syntactic* structure (the text!) with the *visual* structure (the image!)

Onward!

In [13]:
```python
path = "./intro_first"          # any path to any folder
result = list(os.walk(path))    # this will "walk" all of the subfolders and files

print(f"{len(result)}")      # try c:/  (it took my machine 12.7 seconds!)
print(f"{result}")
```

```
5
[('./intro_first', ['cs', 'sci'], ['.DS_Store', 'nottrue.ipynb']), ('./intro_first/cs',
[], ['.DS_Store', 'file35.txt', 'file181y.txt']), ('./intro_first/sci', ['50', '10'], ['.D
S_Store']), ('./intro_first/sci/50', [], ['IDE.txt']), ('./intro_first/sci/10', [], ['IDE.
txt'])]
```

Now, let's incorporate `os.walk` into a series of functions...

In [14]:
```python
#
# Steppingstone, Version 1: call os.walk, return length, optionally print
#

import os
import os.path

def file_walker(path):
    """ starting from the input, named path

        this function "walks" the whole path, including subfolders
        and then... explores any questions we might want :)
        call, for example, with    file_walker("./intro_first")
    """
    result = list(os.walk(path))     # perhaps try w/o converting to a list...
    # print(f"{len(result) = }")
    # print(f"{result = }")
    num_folders = len(result)        # the len is the number of folders...
    return num_folders

#
# when discovering, keep your data close (and your functions closer!)
#
if True:
    """ overall script that runs examples """
    print(f"[[ Start! ]]\n")     # sign on

    path = "./intro_first"       # Remember: . means the current directory
    result = file_walker(path)   # Run!

    print(f"result = {result}")  # Yay for f-strings!

    print("\n[[ Fin. ]]")        # sign off
```

```
[[ Start! ]]

result = 5

[[ Fin. ]]
```

Ok! But we didn't actually "walk" the folders -- or files!

That is, we only counted, and didn't consider, each one...

Let's print all of the folder names!

In [16]:

```
#
# Steppingstone, Version 2: print all of the folder names!
#

import os
import os.path

def file_walker(path):
    """ starting from the input, named path

        this function "walks" the whole path, including subfolders
        and then... explores any questions we might want :)

        call, for example, with    file_walker("./intro_first")
    """
    result = list(os.walk(path))     # perhaps try w/o converting to a list...

    for folder_tuple in result:
        currentpath, subfolders, files = folder_tuple  # always three items, always these.
        print(f"{currentpath}")   # try non-f printing: it's worse!

    num_folders = len(result)        # the len is the number of currentpaths...
    return num_folders

#
# when discovering, keep your data close (and your functions closer!)
#
if True:
    """ overall script that runs examples """
    print(f"[[ Start! ]]\n")     # sign on

    path = "./intro_first"       # Remember: . means the current directory
    result = file_walker(path)   # Run!

    print(f"result = {result}")  # Yay for f-strings!

    print("\n[[ Fin. ]]")        # sign off
```

```
[[ Start! ]]

./intro_first
./intro_first/cs
./intro_first/sci
./intro_first/sci/50
./intro_first/sci/10
result = 5

[[ Fin. ]]
```

If you're on Windows, you likely see some "hidden MACOSX" directories, `__MACOSX`

Task!

Change the above code so that it *skips* any path that contains the string `__MACOSX`

We'll do this together...

But, we must *walk the files*!

Let's print all their full filenames (the full paths)!

In [18]:

```
#
# Steppingstone, Version 3: walk all of the files, printing each one's fullpath
#

import os
import os.path

def file_walker(path):
    """ starting from the input, named path

        this function "walks" the whole path, including subfolders
        and then... explores any questions we might want :)

        call, for example, with    file_walker("./intro_first")
    """
    result = list(os.walk(path))     # perhaps try w/o converting to a list...

    for folder_tuple in result:
        currentpath, subfolders, files = folder_tuple  # always three items, always these.

        if '__MACOSX' in currentpath: continue         # skip the rest of _this_ loop iter

        print(f"{currentpath  }")

        for file in files:       # remember, files is a list of filenames!
            fullpath = currentpath + "/" + file           # construct the full path, or, b
            print(f"   {fullpath  }")
            #contents = GET_STRING_FROM_FILE(fullpath)     # use the fullpath!
            #print(f"{contents[0:42] = }")

    num_folders = len(result)        # the len is the number of currentpaths...
    return num_folders

#
# when discovering, keep your data close (and your functions closer!)
```

```
#
if True:
    """ overall script that runs examples """
    print(f"[[ Start! ]]\n")     # sign on

    path = "./intro_first"       # Remember: . means the current directory
    result = file_walker(path)   # Run!

    print(f"result = {result}")  # Yay for f-strings!

    print("\n[[ Fin. ]]")        # sign off
```

```
[[ Start! ]]

./intro_first
   ./intro_first/.DS_Store
   ./intro_first/nottrue.ipynb
./intro_first/cs
   ./intro_first/cs/.DS_Store
   ./intro_first/cs/file35.txt
   ./intro_first/cs/file181y.txt
./intro_first/sci
   ./intro_first/sci/.DS_Store
./intro_first/sci/50
   ./intro_first/sci/50/IDE.txt
./intro_first/sci/10
   ./intro_first/sci/10/IDE.txt
result = 5

[[ Fin. ]]
```

It's possible to assemble paths using the operating system's "correct" character:

In [19]:
```
os.path.join("/root/Users/secret_stuff/folder_name" , "file_name")
```

Out[19]:
```
'/root/Users/secret_stuff/folder_name/file_name'
```

Since Python is happy with `/` we'll use that for now.

But, we want to get all of the files' *contents* !

Notice there are lots of files named `".DS_Store"` ...

They are binary data used by MacOS - they will cause trouble!

Let's see the trouble, and then fix it:

In [22]:
```
#
# Steppingstone, Version 4: walk all of the files, printing (bits of) each one's contents!
#

import os
import os.path

def file_walker(path):
    """ starting from the input, named path

        this function "walks" the whole path, including subfolders
        and then... explores any questions we might want :)

        call, for example, with    file_walker("./intro_first")
    """
    result = list(os.walk(path))     # perhaps try w/o converting to a list...
    for folder_tuple in result:
        currentpath, subfolders, files = folder_tuple  # always three items, always these.
        if '__MACOSX' in currentpath:  continue
        print(f"{currentpath  }")

        for file in files:       # remember, files is a list of filenames!
            fullpath = currentpath + "/" + file           # construct the full path, or, b
            print(f"   {fullpath  }")
            contents = GET_STRING_FROM_FILE(fullpath)     # use the fullpath!
            print(f"   {contents[0:42]  }")

    num_folders = len(result)        # the len is the number of currentpaths...
    return num_folders

#
# when discovering, keep your data close (and your functions closer!)
#
if True:
    """ overall script that runs examples """
    print(f"[[ Start! ]]\n")     # sign on

    path = "./intro_first"       # Remember: . means the current directory
    result = file_walker(path)   # Run!

    print(f"result = {result}")  # Yay for f-strings!

    print("\n[[ Fin. ]]")        # sign off
```

```
[[ Start! ]]

./intro_first
   ./intro_first/.DS_Store
```

```
---------------------------------------------------------------------------
UnicodeDecodeError                        Traceback (most recent call last)
/var/folders/j2/qb22mym15dg1pw42kg9hl8y00000gn/T/ipykernel_37331/3556423664.py in <module>
     38 
     39     path = "./intro_first"       # Remember: . means the current directory
---> 40     result = file_walker(path)   # Run!
     41 
     42     print(f"result = {result}")  # Yay for f-strings!

/var/folders/j2/qb22mym15dg1pw42kg9hl8y00000gn/T/ipykernel_37331/3556423664.py in file_wal
ker(path)
     24             fullpath = currentpath + "/" + file           # construct the full pat
h, or, better: os.path.join(currentpath,file)
     25             print(f"   {fullpath  }")
---> 26             contents = GET_STRING_FROM_FILE(fullpath)     # use the fullpath!
     27             print(f"   {contents[0:42]  }")
     28 

/var/folders/j2/qb22mym15dg1pw42kg9hl8y00000gn/T/ipykernel_37331/364306208.py in GET_STRIN
G_FROM_FILE(filename_as_string)
     10         # the encoding below is a common default, but not universal...
     11         file_object = open(filename_as_string, "r", encoding='utf-8')    # open!
(Other encodings: 'latin-1', 'utf-16', 'utf-32')
---> 12         file_data = file_object.read()                                   # and get
all its contents
     13         file_object.close()                                              # close t
he file (optional)
     14         #print(DATA)                                                      # if we w
ant to see it

/opt/anaconda3/lib/python3.7/codecs.py in decode(self, input, final)
    320         # decode input (taking the buffer into account)
    321         data = self.buffer + input
--> 322         (result, consumed) = self._buffer_decode(data, self.errors, final)
    323         # keep undecoded input until the next call
    324         self.buffer = data[consumed:]

UnicodeDecodeError: 'utf-8' codec can't decode byte 0xba in position 95: invalid start byt
e
```

The encoding was wrong!

(Those `.DS_store` files are binary, not human-readable.)

We could change to, say, latin-1 and see the bytes. But, let's not...

We really just want to *algorithmically* skip over those files. Let's try it:

Task to try

Add an if statement in the above "steppingstone function" in order to simply skip over any file that begins with a dot (a period character: `"."`)

Then, run it again. (It's ok to leave those dot files' pathnames - or not...)

Two examples leading into our "rolodex" challenges!

Example 1 Let's count how many `.txt` files we have...

In [24]:
```
#
# Rolodex lead-in, example1: counting the number of .txt files...
#
```

```python
import os
import os.path

def file_walker(path):
    """ starting from the input, named path

        this function "walks" the whole path, including subfolders
        and then... explores any questions we might want :)

        call, for example, with    file_walker("./intro_first")
    """
    result = list(os.walk(path))     # perhaps try w/o converting to a list...

    count_txt = 0    # keep count of our .txt files

    for folder_tuple in result:
        currentpath, subfolders, files = folder_tuple  # always three items, always these.
        if '__MACOSX' in currentpath:  continue
        print(f"{currentpath }")

        for file in files:       # remember, files is a list of filenames!
            fullpath = currentpath + "/" + file           # construct the full path, or, b
            print(f"   {fullpath }")
            if file[0] == ".": continue      # skip files that start with dot
            if file[-4:] == ".txt":
                print("Found a .txt file! Adding one...")
                count_txt += 1
            #contents = GET_STRING_FROM_FILE(fullpath)     # use the fullpath!
            #print(f"   {contents[0:42] = }")

    return count_txt   # phew, we're finally returning something else!

#
# when discovering, keep your data close (and your functions closer!)
#
if True:
    """ overall script that runs examples """
    print(f"[[ Start! ]]\n")     # sign on

    path = "./intro_first"       # Remember: . means the current directory
    result = file_walker(path)   # Run!

    print(f"num txt files = {result}")  # Yay for f-strings!

    print("\n[[ Fin. ]]")        # sign off
```

```
[[ Start! ]]

./intro_first
   ./intro_first/.DS_Store
   ./intro_first/nottrue.ipynb
./intro_first/cs
   ./intro_first/cs/.DS_Store
   ./intro_first/cs/file35.txt
Found a .txt file! Adding one...
   ./intro_first/cs/file181y.txt
Found a .txt file! Adding one...
./intro_first/sci
   ./intro_first/sci/.DS_Store
./intro_first/sci/50
   ./intro_first/sci/50/IDE.txt
Found a .txt file! Adding one...
./intro_first/sci/10
   ./intro_first/sci/10/IDE.txt
Found a .txt file! Adding one...
num txt files = 4

[[ Fin. ]]
```

This is an example of a short (1-3 sentence) markdown cell, giving interpretation and context for the above result...

Number of `.txt` files

- It seems that this folder, `intro_first` has four (4) `.txt` files. This seems reasonable!
- We could go further and see what *percentage* of files are `.txt` ...
- It's also worth noting that we're trusting the file extension `.txt` here: some text files could be masquerading as other things... 😊 ?!

The key idea is to ...

- share the results found, contextualized for us sapiens ...
- consider what else could be done, even if we're not doing so ...
- note possible incompletenesses, countervailing forces, concerns in general ...
- use emojis 🦔 !

Second example leading into our "rolodex" challenges!

Example 2 Let's count how many of the `.txt` files contain the substring `'CS'`

In [26]:
```python
#
# Rolodex lead-in, example2: counting the number of .txt files containing 'CS' ...
#

import os
import os.path

def file_walker(path):
    """ starting from the input, named path

        this function "walks" the whole path, including subfolders
        and then... explores any questions we might want :)

        call, for example, with    file_walker("./intro_first")
    """
    result = list(os.walk(path))     # perhaps try w/o converting to a list...
    count_txt = 0    # keep count of our .txt files
    count_CS = 0     # keep count of 'CS' substrings found

    for folder_tuple in result:
        currentpath, subfolders, files = folder_tuple  # always three items, always these.
        if '__MACOSX' in currentpath:  continue
        print(f"{currentpath  }")

        for file in files:       # remember, files is a list of filenames!
            fullpath = currentpath + "/" + file           # construct the full path, or, b
            print(f"   {fullpath  }")
            if file[0] == ".": continue      # skip files that start with dot

            if file[-4:] == ".txt":
                # print("Found a .txt file! Adding one...")
                count_txt += 1
                contents = GET_STRING_FROM_FILE(fullpath)     # use the fullpath!
                if 'CS' in contents:
                    print("Found a 'CS' ... adding 1    (aka 2-True)")
                    count_CS += 1
                # print(f"   {contents[0:42] = }")

    return count_CS, count_txt   # oooh... we can return two things!

#
# when discovering, keep your data close (and your functions closer!)
#
if True:
    """ overall script that runs examples """
    print(f"[[ Start! ]]\n")     # sign on

    path = "./intro_first"       # Remember: . means the current directory
    result = file_walker(path)   # Run!

    count_CS, count_txt = result
    print()
    print(f"num txt files       = {count_txt}")
    print(f"num containing CS   = {count_CS}")
    perc = count_CS*100/count_txt
    print(f"for a CS percentage of {perc:5.2f}%")   # :5.2f means width of 5, 2 dec. place

    print("\n[[ Fin. ]]")        # sign off
```

```
[[ Start! ]]

./intro_first
   ./intro_first/.DS_Store
   ./intro_first/nottrue.ipynb
./intro_first/cs
   ./intro_first/cs/.DS_Store
   ./intro_first/cs/file35.txt
Found a 'CS' ... adding 1    (aka 2-True)
   ./intro_first/cs/file181y.txt
Found a 'CS' ... adding 1    (aka 2-True)
./intro_first/sci
   ./intro_first/sci/.DS_Store
./intro_first/sci/50
   ./intro_first/sci/50/IDE.txt
./intro_first/sci/10
   ./intro_first/sci/10/IDE.txt

num txt files       = 4
num containing CS   = 2
for a CS percentage of 50.00%

[[ Fin. ]]
```

Results:

Number of `CS`-content `.txt` files

- It seems that this folder, `intro_first` has two `'CS'`-containing `.txt` files, out of four total `.txt` files, that is, `50%`

- Reflection: *This seems computationally balanced.*

- Opportunities: We could go further and try this in larger folders - such as this whole machine! Or, we could look for other things (like phone numbers or names in various formats). Or, really, we could ask-and-answer almost any algorithmic question about any subset of files on any machine at all...

- *The fox knows many things, but the hedgehog knows one big thing.* - Archilochus 🦔

Trying other directories/folders

The `path` can be *any* folder on your local machine, allowing for *arbitrary* local exploration and discovery...

For example, **how many files** do I have on this machine?

```
In [28]:
path = "./intro_first"          # any path to any folder?!

path = "C:/"                    # let's use C:/  on windows  (you could use "/" on MacOS)
result = list(os.walk(path))    # this will "walk" all of the subfolders and files

print(f"{len(result)  }")      # this took my machine 12.7 seconds!
#print(f"{result = }")          # let's _not_ print it out...
```

```
0
```

Your task: The Rolodex challenge!

- Here is the homework page that describes hw1's challenges...
- A few questions are "our" design
- Then, ask-and-answer more are of *your* design
- And, you'll answer *your* questions from at least two other "root" directories (the `path` that gets everything started is sometimes called the "root": the folder whose files your functions *walk*! :)
- Create a short **Results:** section after each of "our" and *your* questions. Feel free to use the template above.
- Good luck, walking *far more* than 1,000 files!