import logging as log

print “A short introduction by Daniel Wooten”

~~~
def outline( num_sec , titles ):

    ``` This presentation will go over
          why log?
          how do I log?
          the basics
          formatting
          output to file
          custom instances
          conclusions ’’’

    return( enlightened_audience )
~~~

Why would I bother?

- As a developer
    - Because you don’t hate your users
        - Hint* they will probably include yourself
    - Events might/will occur which should be noticed
    - Can provide more insight into source code than comments
- As a coder
    - Debuggers scare you
    - It’s not the code, it’s the math
    - I don’t have time for steps!

The basics

- logging has 6 basic levels [critical , error , warning , info , debug , notset]
  - These are associated with [50 , 40 , 30 , 20 , 10 , 0]
- Log.basicConfig(level = [value])
  - will set the default level for the given session
  - Anything with a value equal to or greater will be reported, warning is default
- Log.[level_name](‘Message to print’) is the most basic command
- See log_pres_1.py

Formatting

- To save some computational time you may want to wrap with
  - If log.isEnabledfor(value):

    Log.whathaveyou(“This will only be evaluated if the above if statement passes”)
- The previous outputs were ugly and hard to read
  - Fix this with log_instance.basicConfig(args , format = ‘’ , args)
    - format = “standard python string formatting syntax” albeit it uses %(<dictionary key>)s substitution
- .basicConfig is only taken once for each instance, any repeat is ignored
- See log_pres_3.py

Output

- Anytime you're using a logger you're* running a script
  - So you probably don't want your logger output going straight to stdout
- For the basic (root) instance of log you can specify a file (and write type) for the output to be re-directed to
  - This is accomplished with the filename and filemode arguments inside of .basicConfig, simply specifying filename will redirect your output
  - You can direct different levels of output to different files, or even some to the screen and to a file, more about this in a moment
- See log_pres_4.py

*you should be

Custom Instances

- The strangest part of the logger utility, various methods only work for EITHER the root logger or a created instance, not both
- Why have custom instances
  - For specific modules such that the info they return is different from the calling environment
  - For different parts of a single module
  - To enable different output paths for different messages
- Keywords to know, handlers and formatters
- You can only modify custom instances through handlers

Creating a custom instance

- mat_log = log.getLogger(‘matrix_logger’)
- mat_log.setLevel(value)
    - will set the default for the mat_log logger

Modifying any instance

- Create a handler
  - name = log.FileHandler(filename = ‘handler_out.txt’ , mode = ‘w’)
  - This is the real power of handlers, we can specify, by their creation, where their output goes
- Set handler level
  - name.setLevel([value])
- Set handler format
  - Formatter = log.Formatter(‘format string’)
  - name.setFormatter(formatter)
- Attach handler to logger instance
  - Instance.addHandler(name)
- See log_pres_5.py

Things we didn’t talk about

- The known-unknowns
  - Filters
    - More advanced options than just [level] to filter messages by
  - Various other output methods including ones triggered by events in code and the actual system time
  - Integration with the warning package in python
- The unknown-unknowns
  - ∞

Conclusions

- Logging can make your life, and you user’s lives, less stressful
- Logging can smooth the development cycle
- If you want to do it, most likely the logger can…
    - Print messages to screen
    - Print messages to file
    - Print different messages to different files
    - Print different messages to different output types
    - Help you to avoid using pdb at all costs
    - Give you an excuse to debug with print statements

Where do I start?

- https://www.google.com/?gws_rd=ssl#q=python+logger
- Seriously?
  - Why you’re logging will determine the functionality that you need
    - Your own development purposes : simple output to a file
    - Collaborative work : maybe output to a file, but only for the parts you wrote
    - Development : some outputs to screen, others to file, user inputs
    - Development/stewardship : intricate hierarchy of reporting and event logging
- See log_pres_1.py