Code Snippets to accompany slides for A Dinosaur and a Python Walk Into a Bar

RBLandau 20210115

```
############ BEGINNER DECORATOR EXAMPLE #############

def my_decorator(func):
    def wrapper():
        print("Before the function is called.")
        result = func()
        print("After the function is called.")
        return result
    return wrapper


use as

@my_decorator
def add(a, b):
    return a+b
```

```
############ TIMESTAMP #############

import datetime
 . . .
def fnsGetTimestamp(self):
    '''Return timestamp with or without milliseconds.
    '''
    if self.bTimeHires:
        return datetime.now().strftime('%Y%m%d_%H%M%S.%f')[:-3]
    else:
        return datetime.now().strftime('%Y%m%d_%H%M%S')

==>
    20201103_153703.284
  or
    20201103_153703
```

```
######### ENVIRONMENT VARIABLE CONTROL ###########

# bash command in the shell:
    export some_env_var=some_value
# e.g.,
    export TRACE_LEVEL=3
# and restart the program.

try:
    tracelevel = int(os.getenv("TRACE_LEVEL", defaultlevel))
except ValueError:  # Take default if not int
    tracelevel = int(defaultlevel)
```

Beware: the value returned from getenv() is always a string even if it looks like an int

(more)

Currently implemented:

TRACE_LEVEL	level of detail
TRACE_TARGET	standard and/or HTML
TRACE_FILE	filename
TRACE_FACIL	facility codes to trace or not
TRACE_TIME	seconds or milliseconds
TRACE_PRODUCTION	turn off all tracing

```
############ LOOKING AT OUTPUT #############

export TRACE_LEVEL=3
python whateverprogram.py  2>&1  | less


'less' utility makes it easy to scroll and search in output stream
```

```
######### IF PRODUCTION MODE ###########

# If in PRODUCTION mode, skip over all the printing.
    def ntrace(self, level, line):
        if (not self.isProduction() or level == 0):
            # (real trace code)
         . . .
        else:       # If in production mode and level > 0
            pass    #  do nothing.
```

```
# Same for the decorator definitions.
    if NTRC.isProduction():
        def ntrace(func):
            return func     # Null decorator, does nothing.
    else:
        def ntrace(func):
            @wraps(func)
            def wrapper(*args,**kwargs):)
             . . .
            return wrapper
```

```
export TRACE_PRODUCTION=YES
```

```
######### IMPORTS ###########

# Import module to get singleton instance and decorator functions

from NewTrace import NTRC, ntrace, ntracef
```

```
############ SINGLETON INSTANCE OF CLASS #############

# Make a singleton of the NewTrace instance (from ActivePython).
class Singleton(type):
    _instances = {}
    def __call__(cls,*args,**kwargs):
        if cls not in cls._instances:
            cls._instances[cls] = super(Singleton,cls).__call__(*args,**kwargs)
            cls._provenance = "singleton instance of class %s" % cls._instances[cls]
        return cls._instances[cls]

class CSingletonNewTrace(CNewTrace):
    __metaclass__ = Singleton
    _whatsit = "singleton instance of class NewTrace"

# NEW VERSION NTRC INSTANCE
NTRC = CSingletonNewTrace()
```

```
############ SPECIFIC INSERTED TRACE CALLS ############


# Simple.  Just like sprinkling print() but more informative.
    NTRC.ntracef(3, "READ", "proc fdGetParams1 file not found |%s|" % (mysFile))
==>
20210107_222604 3 READ  proc fdGetParams1 file not found |../hl/servers.csv|

or

NTRC.ntracef(3, "FMT", "proc FormatQuery item key|%s| val|%s| result|%s|"
    % (sAttrib, sValue, result))
==>
20210108_130214 3 FMT   proc FormatQuery item key|nDocSize| val|50| result|50|



(more)
```

```
--------------------------------------------------------------------------------

# Less simple: contains a lot of information useful for debugging.
        NTRC.ntracef(3, "SERV", "proc mAddDocument serv|%s| id|%s| "
            "docid|%s| size|%s| assigned to shelfid|%s| remaining|%s|"
            % (self.sName, self.ID, mysDocID, cDoc.nSize, sShelfID,
            cShelf.nFreeSpace))
# Note continued string on separate lines (easy with quotes), and
#  continued tuple of values (easy with parens).

# Yes, "string % (values)" rather than f-strings because
#  1.  Python 2 when starting out;
#  2.  I think it's actually more readable;
#  3.  Dinosaur.
==>
20210107_222605 3 SERV  proc mAddDocument serv|Desperate Backup 11| id|V1| docid|D2|
size|50| assigned to shelfid|H01| remaining|9999900|
```

(more)

```
--------------------------------------------------------------------------------

# Priority level 0 can be used for lines that should *always* print,
#  even in production mode.
20210107_222604 0 MAIN  proc Document Preservation simulation
 . . .
20210107_222608 0 MAIN  proc End time stats: wall|   4.011| cpu|   0.188|
######### DECORATOR VERSIONS ###########

# Vanilla version of function call with trace decorator.
@ntrace
def main(mysInputFilename):
==>
20210107_222604 1       entr main args=('params.txt'),kw={}
 .  .  .
20210107_222606 1       exit main result|None|
--------------------------------------------------------------------------------

# Chocolate function call with facility name that can be filtered or searched.
class CServer(object):
 . . .
    @ntracef("SERV")
    def __init__(self,mysName,mynQual,mynShelfSize):
==>
20210107_222604 1 SERV  entr __init__ <cls=CServer id=||> |('Desperate Backup 11', 1,
10)| kw={}
 . . .
20210107_222604 1 SERV  exit __init__ <cls=CServer id=|V1|> result|None|
# Note that class instance ID is listed on exit.

(more)
```

```
-----------------------------------------------------------------------------

# Tutti-fruity version of function call.
@ntracef("UTIL", level=5)
def fnsGetTimeStamp():
 . . .
```

```
--------------------------------------------------------------------------------

# Can even declare multiple facilities for filtering later.
# Facility code "SHOW" might be for things that I really, really want to see
#  when I've filtered out everything else.
    @ntracef("SHOW")
    @ntracef("SERV")
    def __init__(self,mysName,mynQual,mynShelfSize):
==>
20210107_222604 1 SHOW  entr __init__ <cls=CServer id=||> |('Desperate Backup 11', 1, 
10)| kw={}
20210107_222604 1 SERV  entr __init__ <cls=CServer id=||> |('Desperate Backup 11', 1, 
10)| kw={}
 .  .  .
20210107_222604 1 SERV  exit __init__ <cls=CServer id=|V1|> result|None|
20210107_222604 1 SHOW  exit __init__ <cls=CServer id=|V1|> result|None|
-------------------------------------------------------------------------------

# Show return result on exit from function.
@ntracef("READ")
def fdGetParams(sFile, lGuide):
==>
20210107_222604 1 READ  entr fdGetParams args=('../hl/installtest/clients.csv',
['Institution', ['Collection', 'Quality', 'Count']]),kw={}
 . . .
20210107_222604 1 READ  exit fdGetParams result|{'MIT': [['Mags', 1, 10]]}|

or

@ntrace
def fnbValidateDir(sPath):
==>
20210108_130214 1       entr fnbValidateDir args=('../hl/a0',),kw={}
 . . .
20210108_130214 1       exit fnbValidateDir result|True|
--------------------------------------------------------------------------------

# Easy to follow logic in the ntrace listing.
# Mainly entries and exits, with a few added trace lines.
# Note that the same function names may appear in different facilities (modules).
20210107_222606 1 SERV  entr mAddDocument <cls=CServer id=|V1|> |('D9782', 'T1')|
kw={}
20210107_222606 1 SHLF  entr mAcceptDocument <cls=CShelf id=|H01|> |('D9782', 50,
'T1')| kw={}
20210107_222606 1 SHLF  entr mAddDocument <cls=CShelf id=|H01|> |('D9782', 'T1')|
kw={}
20210107_222606 1 COPY  entr __init__ <cls=CCopy id=||> |('D9782', 'T1', 'V1')| kw={}
20210107_222606 1 COPY  exit __init__ <cls=CCopy id=|X9782|> result|None|
20210107_222606 3 SHLF  proc mAddDocument made copy|X9782| of doc|D9782| from
client|T1|
20210107_222606 1 COPY  entr mShelveCopy <cls=CCopy id=|X9782|> |('V1', 'H01',
489051, 489100)| kw={}
20210107_222606 1 COPY  exit mShelveCopy <cls=CCopy id=|X9782|>
result|X9782+V1+H01+[489051,489100]|
20210107_222606 1 DOC   entr mCopyPlacedOnServer <cls=CDocument id=|D9782|>
|('X9782', 'V1')| kw={}
20210107_222606 1 DOC   exit mCopyPlacedOnServer <cls=CDocument id=|D9782|>
result|D9782+X9782+V1|
20210107_222606 1 SHLF  exit mAddDocument <cls=CShelf id=|H01|>
result|V1+H01+D9782+X9782|
20210107_222606 1 SHLF  exit mAcceptDocument <cls=CShelf id=|H01|> result|True|
20210107_222606 3 SERV  proc mAddDocument serv|Desperate Backup 11| id|V1|
docid|D9782| size|50| assigned to shelfid|H01| remaining|9510900|
20210107_222606 1 SERV  exit mAddDocument <cls=CServer id=|V1|> result|V1+H01+D9782|
--------------------------------------------------------------------------------

# All comes out in the same millisecond.  Processing impact light.
20210108_183424.018 1 SERV  entr mAddDocument <cls=CServer id=|V1|> |('D9782', 'T1')| 
kw={}
20210108_183424.018 1 SHLF  entr mAcceptDocument <cls=CShelf id=|H01|> |('D9782', 50, 
'T1')| kw={}
20210108_183424.018 1 SHLF  entr mAddDocument <cls=CShelf id=|H01|> |('D9782', 'T1')| 
kw={}
20210108_183424.018 1 COPY  entr __init__ <cls=CCopy id=||> |('D9782', 'T1', 'V1')| 
kw={}
20210108_183424.018 1 COPY  exit __init__ <cls=CCopy id=|X9782|> result|None|
20210108_183424.018 3 SHLF  proc mAddDocument made copy|X9782| of doc|D9782| from 
client|T1|
20210108_183424.018 1 COPY  entr mShelveCopy <cls=CCopy id=|X9782|> |('V1', 'H01', 
489051, 489100)| kw={}
20210108_183424.018 1 COPY  exit mShelveCopy <cls=CCopy id=|X9782|> 
result|X9782+V1+H01+[489051,489100]|
20210108_183424.018 1 DOC   entr mCopyPlacedOnServer <cls=CDocument id=|D9782|> 
|('X9782', 'V1')| kw={}
20210108_183424.018 1 DOC   exit mCopyPlacedOnServer <cls=CDocument id=|D9782|> 
result|D9782+X9782+V1|
20210108_183424.018 1 SHLF  exit mAddDocument <cls=CShelf id=|H01|> 
result|V1+H01+D9782+X9782|
20210108_183424.018 1 SHLF  exit mAcceptDocument <cls=CShelf id=|H01|> result|True|
20210108_183424.018 3 SERV  proc mAddDocument serv|Desperate Backup 11| id|V1| 
docid|D9782| size|50| assigned to shelfid|H01| remaining|9510900|
20210108_183424.018 1 SERV  exit mAddDocument <cls=CServer id=|V1|> 
result|V1+H01+D9782|
######### INSTANCE IDENTIFIERS ###########

# When you have many instances, it is much easier to pass string identifiers
#  rather than instances (addresses).
# Note use of string identifiers for instances rather than actual instance pointers.
# Takes little time (one dictionary lookup, not a big deal) but saves lives.
20210107_222605 1 DOC   entr __init__ <cls=CDocument id=||> |(50, 'T1', 'C1')| kw={}
20210107_222605 3 DOC   proc init client|T1| created doc|D9706| size|50|
20210107_222605 1 DOC   exit __init__ <cls=CDocument id=|D9706|> result|None|

--------------------------------------------------------------------------------

# Create id attribute in instance's __init__().
20210107_222605 1 COPY  entr __init__ <cls=CCopy id=||> |('D17', 'T1', 'V3')| kw={}
 . . .
20210107_222605 1 COPY  exit __init__ <cls=CCopy id=|X20017|> result|None|


(more)

--------------------------------------------------------------------------------

# Cheap way to make unique IDs for class instances.
import  itertools
    . . .
    #<global to the class>
    # Note: getID() calls next() on the itertools count() function.
    getID = itertools.count(1).next
        . . .
        #<in __init__>
        self.ID = "V" + str(self.getID())

# yields a stream of IDs: V1, V2, V3, . . .
```

```
--------------------------------------------------------------------------------

# Store the ID in a dictionary that translates to the instance
#<in __init__>
    self.ID = "D" + str(self.getID())
    dID2Document[self.ID] = self

# And get the instance back from the ID.
    cDoc = dID2Document[sDocID]


--------------------------------------------------------------------------------

# Sort a dictionary of IDs by number
#  so you get [A1, A2, A11, A12] instead of [A1, A11, A12, A2].
@ntracef("UTIL")
def fnSortIDDict(dIn):
    '''
    Sort a dictionary with keys of the form <letter><number>.
    Return a tuple of item tuples from the dict in numeric key order.
    (Readable code rather than unmaintainable one-liner.)
    '''
    lTmp1 = ((fnIntPlease(x[0][1:]), x) for x in dIn.items())
    lTmp2 = sorted(lTmp1, key=lambda y: y[0])
    lTmp3 = (z[1] for z in lTmp2)
    return tuple(lTmp3)
```

```
--------------------------------------------------------------------------------

# Log input and processing parameters for the run
20210107_222604 MAIN INFO - Simulation parameters
20210107_222604 MAIN INFO - Command line|['main.py', '../hl', 'installtest', '0',
'1', '--lifek=693147', '--ncopies=1', '--audit=0', '--ndocuments=10000']|
20210107_222604 MAIN INFO - Usable CLI line|python2 main.py ../hl installtest 0 1 --
lifek=693147 --ncopies=1 --audit=0 --ndocuments=10000|
20210107_222604 PARAMS INFO - familydir|../hl| specificdir|installtest|
20210107_222604 PARAMS INFO - RANDOM random seed|1|
20210107_222604 PARAMS INFO - begin simulation timelimit|100000|hr=|10|metricyr
defaultlimit|100000| hr=|10|metricyr
20210107_222604 PARAMS INFO - POLITE time|1000|msec
20210107_222604 PARAMS INFO - LOG    logfile|-| loglevel|INFO|
20210107_222604 PARAMS INFO - TRACE  traceproduction|False|
20210107_222604 PARAMS INFO - CLIENT client|MIT| collection|Mags| quality|1|
ndocs|10|
20210107_222604 PARAMS INFO - ALLCLIENTS nDocuments|10000| override if nz
```