[redacted]

July 8, 2010

By email to rule-comments@sec.gov

Ms. Elizabeth M. Murphy
Secretary
Securities and Exchange Commission
100 F Street, N.E.
Washington, D.C. 20549 Re:

Re: **File No. S7-08-10 Asset-Backed Securities**

Dear Ms. Murphy:

I write to comment on aspects of Release Nos. 33-9117; 34-61858 regarding the use of XML data files and the waterfall computer program in the offering disclosure, and reporting for asset-backed securities. I am a securitization attorney with experience since 1990 in residential and commercial mortgage backed, credit card, auto loan and stranded asset transactions. From 1998 until 2008 I was in-house counsel to Washington Mutual, Inc. where I also was responsible for the initial organization of its fixed-income broker-dealer subsidiary, WaMu Capital Corp. My comments will be primarily based on my experience in residential assets.[1]

Endorsement of the Framework

The Commission's proposals to standardize data fields and presentation format and to require the integration of numeric and narrative descriptions of cashflows will improve disclosure, monitoring and regulation of securitizations and encourage enhanced attention to detail by sponsors and underwriters. It will close the loop between the predominantly numeric orientation by the designers of cashflows and the bond administrators responsible for distribution calculations, on the one hand, and lawyers drafting contractual provisions, on the other. The common ground provided by the waterfall computer program's expression of the rules will better confirm the business intent and further clarify their contractual expression in the operative documents.

Scope of the Waterfall Computer Program

The scope of the program should be limited. It should not reproduce in full the exact working of the distribution rules under every contingency. Issues sometimes arise in the administration of distributions that require considerable interpretation and judgment to make the precise amounts to each class of security accurate. Transaction operative documents contain extensive provision for possibilities that seldom or never arise in practice. If all of these contingencies are included in the disclosure, waterfall computer program users would have to make many assumptions about the future state of pool assets that create them. Since the effect on the magnitude of distributions are small except for structures with highly leveraged features, the contingencies should be

[1] Readers interested in an informal presentation of these and other issues in the proposing release will find commentary at http://blog.revisedregab.com and http://www.pylaw.org.

disregarded as second- or third-order effects. A model derives its usefulness from abstraction and its ability to demonstrate results under a manageable number of assumptions.

Rather, the program should permit an assessment of the amount outstanding under assumptions of prepayment, default probability and severity given default. This facility would be similar to, but with more user control of, the decrement tables in the prospectus currently and the conclusory analysis represented by ratings. Of course, if the program is not sufficient to cover material contingencies an appropriate disclosure to that effect would be necessary.

Content and Form of Inputs

The data fields that the Commission proposes are not all equally suitable for input to the waterfall computer program. While the contractual terms of the pool assets do enter into the calculation of distributions (beginning principal balance, maturity, interest rate, any adjustment terms, etc.), most of the fields provide information that is useful in estimating default and severity given default probabilities. Those probabilities provide a possible input, at the option of the user, to vary assumptions about realized losses. A data structure for prepayment speeds should also be provided, together with a sample table with stated assumptions against which to test that facility.

Two data fields are problematic if nonpublic borrower information is to be provided. The date on which the loan closed, together with the actual sale price of the residence provide sufficient information in most recording offices to identify the borrower. The date of closing provides no useful information that cannot be captured by the month or quarter of closing. The approximate sale price, similarly, is adequately captured by the loan to value ratio. If these date fields are *not* so modified, the Commissions concerns about disclosing zip code will be defeated. If so, no reason appears not to permit zip code improvement. In any event, the metropolitan area is too coarse a unit of measurement to be useful, and a 3-digit zip code should be permitted.

XML format data provides the benefit of a consistent means of extracting pool asset data without the complications of extraneous formatting currently used to make web versions visually attractive. This is accomplished through the beneficial separation of content from design. Design elements are isolated to a portion of the file (through XSLT style markup that governs the rendering of data on the screen or print), while the XML data remains unencumbered for easy extraction by program such as the waterfall computer program.

The Commission should also consider the benefits of requiring static pool information be provided in XML format.

Facilities for Alternative Inputs and Analysis

While the waterfall computer program provides a base level of functionality, many investors have relied on vendors of proprietary systems to analyze investments. To avoid interfering with their ability to do so, the waterfall computer program requires an application programming interface (API) that specifies the required form of inputs, the input required by each module and the outputs produced.

Underwriters should also have the option to permit the waterfall computer program to be provided in forms such as a spreadsheet[2] or a proprietary input file supplementally as a non-issuer freewriting prospectus.

Outputs

The waterfall computer program should project monthly (or another appropriate interval) receipts of scheduled interest and principal along with prepayments and other reductions in principal on the individual assets in the pool under stated assumptions, such as prepayment speeds, interest rates or default and loss severity given default. Each such assumption should be represented in a way that permits the user to vary the condition. The waterfall computer program should also provide for each trigger in which the rules for distribution change as a result of some condition, such as a clean-up call, a stepdown date or loss or delinquency levels.

It is very important that the waterfall computer program provide for cases in which distributions depend upon the character of the cash received. Examples include shifting of interest rules on prepayments, prepayment penalties allocated to a class, strips on premium loans, advances subject to reimbursement or payments under swap agreements.

The waterfall computer program should include an output that captures and allocates results of realized losses in a way that permits the user to vary the levels of anticipated losses. The resulting periodic payment of principal and interest on each security, according to its respective entitlement, is the primary output.

Narrative

While the waterfall computer program is responsible for translating the logic of the transaction structure to distributions given the pool asset data and assumptions, the narrative is responsible for describing the logic. Except for simple structures, this task requires the extensive use of defined terms, just as the implementing logic is expressed in discrete variables and functions. To keep the narrative both precise and comprehensible, it should be introduced by an overview.

The overview should begin with a description of the source of payments due to the pool assets[3] and the deductions from those payments for compensation to transaction parties such as the servicer. The overview should then discuss how distribution rules change over time. For each fixed or variable period, the overview should discuss the priority of distribution of the available distribution amount and whether distributions among securities of the same priority is proportional to their balances or divided on some other basis. The method of calculating interest for the applicable period should be identified by security, and the rule for application to interest and principal should be described. The overview should then discuss changes to the usual rules

---

[2] A spreadsheet is a program with a graphical user interface that permits the contamination of clean data and programming logic by human error, especially when it includes features designed primarily to make highly decorated output.

[3] This might include payments other than those made by the borrower, such as mortgage insurance, pool insurance or liquidation proceeds.

that are contingent upon the nature of the payments (for example, prepayments and liquidations) or triggers.

Special types of rules, such as shifting of interests, would benefit from a description of their purpose in the structure. Disclosures are easier to follow when the underlying motivation for special features is explicit.

Once an overview is provided, a rigorous discussion of the rules should continue to be the practice. Readers who wish to understand the details in depth will have been oriented by the overview and readers who are not will escape the burden of a lengthy disclosure. Unfortunately, the only way to make disclosure simple is to make structure simple.

Demonstration

To illustrate what a waterfall computer program would look like, in part, the exhibit that follows the letter includes Python source code of a program to generate a decrement table for a recent actual transaction with user specified assumptions about single monthly mortality of individual mortgage loans. This differs from the usual assumption that the pool consists of representative mortgage loans that represent an aggregate of individual loans with the same payment characteristics. The methodology that is applied in the usual prospectus disclosure has an unrealistic feature, which is that only portions of such loans prepay. While partial prepayments, called curtailments, do occur they are much less common than prepayments in full. The example in the exhibit considers each loan in the pool individually and applies a randomization factor to determine if a particular loan pays off in a given period. This is the sort of enhancement that permits greater investor participation in analysis.

The waterfall computer program is in novice style and heavily commented. This is to make it easier to follow for those unfamiliar with Python or programming. It is not intended as an example of best programming practices.

Integration with Operative Documents

To mitigate the tension between the disclosure through the waterfall computer program and the contractual implementation of the intent of the structure, the pooling and servicing agreement or equivalent could include a clause such as the following: "The [responsible transaction party] shall interpret and administer this Agreement to give effect to its terms and conditions governing distributions to securityholders consistently with the computer program in Exhibit __ to the extent that the assumptions described in Exhibit __ conform to the actual conditions for any Distribution Date."

Updating and Reconciliation

Current pool asset data should be provided often enough to permit observers to compare actual with projected results. Only the data fields that change as a result of payment related activity should be updated, although it would be helpful if the fields that speak as of the cut-off date also be included. So, for example, no updated credit scores would be provided but the credit scores at origination (or later if included in the original data tape) would be.

Actual results will differ from projected results, and interested parties monitoring performance would benefit if the responsible transaction party provided a quarterly reconciliation

to show differences between an assumed cut-off date set of assumptions for the waterfall computer program and actual distributions. This could cover differences due to asset pool performance, interest rate changes, the more detailed administrative model and any differences due to unidentified causes.

Phase-in

Underwriters, issuers, parties responsible for bond administrators, investors and other users of the waterfall computer program would benefit from a phase-in period to familiarize themselves thoroughly with the preparation and use of the waterfall computer program, to allow commercial vendors to integrate their offerings with the program and to allow practices to converge to shared conventions. Along with whatever implementation effective date that the Commission determines for this requirement, it would be appropriate during the first year to have the waterfall computer program treated similarly to static pool data disclosure when it was first introduced. It would be deemed to not be part of the registration statement, and good faith errors would not carry strict Section 11 liability.

Importance of Standards

A waterfall computer program can be written in many different ways even within a single programming language.[4] While any single portion of a program may be easy to understand, it can be confusing to have to keep straight several different ways of dealing with the same type of calculation. For example, many structures with subordination use a shifting of interests feature. This rule directs prepayments to the senior class, which increases, or shifts, the relative proportions of the interests of the senior and subordinate classes in the transaction. Usually the amount of prepayments begins at 100% and decreases (steps down) at scheduled intervals provided losses remain below set levels. An example is included in the exhibit.

Because this is a pattern that occurs often, it would benefit from a consistent implementation. This might develop through the practice of issuers conforming disclosure practice to what they see other issuers doing. Another way would be to adapt the open source model of a self-appointed standards board to publish, receive comment and maintain reference modules. This has been a very successful approach in many academic and computer science applications. It could arise spontaneously among securitization participants but more likely will require the sponsorship of a body such as SIFMA or the American Securitization Forum. The willingness of an industry group to undertake this task during a phase-in period would provide strong reason for a phase-in period that requires the waterfall computer program but does not deem it part of the registration statement.

Burden on Drafting and Opining Attorneys

Some securitization attorneys have expressed strong doubts that they would be able to deal with computer code. This is understandable but mistaken. It is not necessary that attorneys be sufficiently skilled to write waterfall computer programs but they do need to be able to read them.

---

[4] For an amusing illustration *see* http://www.99-bottles-of-beer.net/language-python-808.html (Python programs in 10 different idioms to print the lyrics to the popular ditty); *cf.* http://www.99-bottles-of-beer.net/abc.html (other languages).

That is not as difficult as may be imagined. It involves, in abbreviated form, the same skills in parsing, logic and defined terms as drafting. The syntax of Python, for example, is far more consistent and intuitive than that of the *Uniform System of Citation*. Attorneys cannot perform their required roles without becoming literate enough to be able to read and understand the waterfall computer programs.

If the burden of authoritatively defining the structure is shifted to the programmers of the waterfall computer program and the attorney is merely a scrivener, the attorney still needs to be able to read the result to draft the contractual terms. If the definitive expression remains the pooling agreement or equivalent transaction document, the attorney still needs to be able to explain to the programmer how to understand it. For example, many usages that are perfectly intelligible to attorneys, such as clauses using *notwithstanding*, can baffle a programmer. As confident as the attorney is in her explanation, she must still be able to read the result to know if she has been understood.

While opining attorneys are justified in excluding the computational results of the waterfall computer program from the scope of their negative assurance opinions to underwriters, they should undertake to provide comfort on programming logic, excluding syntax matters.

Comment on Specific Questions

Is there an alternative form of required information filing that would be more useful to investors, subject to the limitation that executable code may not be filed on EDGAR?

Potentially. No language should be excluded simply because it is compiled before running rather than interpreted as it is run. Depending on the security measures taken, source code for a waterfall computer program in a compiled language such as C++ is as innocuous or dangerous as source code in an interpreted language, such as Python, PERL or Ruby.

Should we require, as proposed, that the Rule 424(h) filing include the waterfall computer program?

No, it should be in the Rule 430 filing. The waterfall computer program should be a part of the statutory prospectus and available to the investor before a sale. A preliminary prospectus is not, however, necessary for this purpose. The pricing and expense information has never been relevant to investors in asset backed securities and now that asset backed securities will have distinct registration forms the requirement should be dropped. Unlike an equity or debt investor in an operating company who has an interest in knowing how much of its investment will be applied to the enterprise, the asset backed investor has no interest in how the purchase proceeds are allocated among the issuer and underwriters.

Is it appropriate to require issuers to submit the waterfall computer program in a single programming language, such as Python, to give investors the benefit of a standardized process?

Yes. Even if investors have sufficient resources to deal with multiple programming languages, their regulators and other interested parties may not. Lack of a common standard would also hinder comparative analyses of different issuer transactions.

If so, is Python the best choice or are there other open source programming language alternatives (such as PERL) that would be better suited for these purposes?

Each programming language has its relative strengths. Given application requirements, programmers may prefer one over the other. C++ provides advantages in working within the Windows operating environment. PERL, the duct tape of the Internet, is particularly suited for one-off problems that can be solved by quickly adapting existing code to a new use. Java has strengths for graphical user interface applications that will run unmodified on different operating systems. Ruby is concise yet understandable and has widely accepted coding conventions that promote consistency. Haskell enforces referential integrity so that code will produce identical results unaffected by side effects of other code. Lisp is very concise and provides good facilities to program the programming. R has libraries for almost every statistical tool that exists. C is fast, universally portable, compact and very mature. Python is highly readable and maintainable, stable and well adapted to programming by snapping together module pieces. Python can embed modules from other languages.

A trained programmer could prepare the waterfall computer program in any of these languages. The compiled languages, C, C++ and Java would be more difficult for novices to follow than the interpreted languages. Among the interpreted languages, Ruby, Haskell and Lisp are somewhat terse compared to Python and, for that reason, may also be more difficult for beginners to read. Python represents a good balance between programmer productivity and user accessibility.

Should more than one programming language be allowed?

On if there is a single required language should additional languages should be allowed. It should be the issuer's choice whether to provide the waterfall computer program in additional languages as part of the prospectus or to rely on the underwriter to provide it as a non-issuer freewriting prospectus.

If so, which ones and why?

Any language, open source or proprietary, should be permitted, based on market demand, because the Commission should encourage an investor to use whatever combination of analytic tools is most effective for that investor.

Should we restrict ourselves to only open source programming languages or allow fully commercial or partly-commercial languages (such as C-Sharp or Java) to be used?

The required language should be open source so that no investor or other interested party is required to pay any fee to a vendor or to incur potential intellectual property infringement liability for using or modifying a waterfall computer program.

Are there other requirements we should impose on the possible computer programming languages that are used to satisfy this requirement, other than that such languages be open source and interpreted?

The Commission should not limit its consideration to interpreted languages. Source codes for compiled and interpreted languages do not require separate treatment to protect the integrity of

the EDGAR system. The types of precautions required for system security are identical whether the code is to be invoked by an interpreter and executed or first by a compiler and then executed.

Would downloading the code onto a local computer give rise to any significant risks for investors? If so, please identify those risks and what steps or measure we should take to address the risks, if any.

The risk posed to investor systems depends primarily on the individual user's sophistication and alertness to security and the effectiveness of the user's organizational information technology protocols. Part of the standards setting process should be identifying the types of system operations that are permissible within the program, those that should be allowed only with an advisory statement and those that should be prohibited.

Should the outputs of the waterfall computer program be specified in detail by rule, or broadly defined to afford flexibility to ABS issuers?

The outputs should be broadly defined to include the amounts and character of each periodic distribution and information on the source of payments for cases in which the source of payment affects amounts or allocations. Also, the outputs should include beginning and ending amounts or levels of variables that affect future payments, such as triggers.

Should we require comments in the code that explain what each line does?

Although line-by-line commenting, like that used in many parts of the attached exhibit, is helpful to novices, it does not add much to the understanding of users who are sufficiently versed in the language. The waterfall computer program and each of its functions, classes or methods, however, should contain a docstring that conforms to standard Python conventions.[5] An extended comment should also be included that explains, in a general way, the correspondence between each major portion of the program and the treatment in the narrative portion of the prospectus.

Is this necessary given the narrative disclosure of the waterfall in the prospectus?

Yes. Understanding the narrative helps in understanding the implementation in the waterfall computer program but does not provide sufficient information to understand how to follow the cashflow through the program or how to modify the program.

Is the proposed requirement to provide the waterfall computer program with the proposed Rule 424(h) prospectus as of the date of filing and a final prospectus under Rule 424(b) as of the date of filing appropriate?

It is appropriate only if the Commission determines that a preliminary prospectus without price and proceeds/expenses information and a final prospectus with price and proceeds/expenses information are still necessary given the introduction of Forms SF-1 and SF-3 and the lack of significance of this information to investors. The proceeds/expenses information has no bearing on the future value of the investment. The price disclosure never varies from the phrase "are being offered by the underwriters from time to time for sale to the public in negotiated transactions or otherwise at varying prices."

---

[5] PEP 257 Docstring Conventions http://www.python.org/dev/peps/pep-0257/

Should we provide a transition period prior to the required compliance date that would allow filers to submit only test filings?

No. Test filings are needed only to confirm that the submission is in the form required to be lodged on the EDGAR system and are non-public. What is needed is a period in which waterfall computer programs can be freely disseminated with offerings to familiarize all participants with their composition and use and to promote the development of standardization. This will only happen if the material is deemed not a part of the registration statement during the transition period.

Is our proposal to permit the filing of an exhibit to disclose additional program functionality appropriate?

Although additional program functionality is permissive, Item 601(b)(105) makes its filing mandatory, rather than permissive.

Are there any impediments that issuers would face if they are required to file the waterfall computer program on EDGAR?

No, filing the program on EDGAR presents no unusual difficulty because it is just text. Provided it is filed as plain text (.txt) rather than formatted as HTML, downloading or copying the waterfall computer program should permit it to be run in the Python interpreter program without any editing. Python treats whitespace (tabs and spaces) as part of the syntax. This makes consistency in indentation important to preserve in the filed version.

We note that the waterfall computer program and the narrative description of the waterfall would need to be accurate and the accuracy of one would not compensate for inaccuracies in the other.

The accuracy desired of the narrative is that it describe the material terms and conditions of the operative agreements. The accuracy of the waterfall computer program is that it implement that description for the required pool asset data given stated assumptions to produce numerically correct results. Because the waterfall computer program should not attempt to fully address every possible contingency that the narrative summarizes the standard should be *accuracy given the same level of detail* and not identical level of detail in coverage of all the same conditions.

XML Tagging of the Waterfall Computer Program

A key feature of the waterfall computer program is that it be able to parse the pool asset file data in XML format. The Commission should not require the waterfall computer program source code itself be required to be submitted as tagged XML data. There is no reason why source code could not be presented as an XML document, but it is not commonly done and no specification exists to provide a common presentation.

The following line is the source code for a minimal Python program:

```
print “Hello, world!”
```

It might look like the following tagged in XML

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="http://www.revisedregab.com/xmlsample.xsl"
type="text/xsl" ?>
<program>
```

```
        <keyword name = “print”>
             <argument>”Hello, world!”</argument>
        </keyword>
    </program>
```

The Commission should reconsider the need for this requirement.

Intellectual Property Rights in the Waterfall Computer Program

To avoid disputes about the use of the waterfall computer program by investors, regulators, vendors and other issuers, the Commission should require that a waterfall computer program should be provided under a royalty free license similar to those commonly provided with open source software.

I hope that the Commission finds these comments useful. Please call me at (941) 907-0645 or email me at info@revisedregab.com with questions if needed.

Very truly yours,

Richard Careaga

Attachment: Demonstration Program Exhibit

```
#!/usr/bin/env python
# encoding: utf-8
"""
demonstration.py

Created on 2010-07-07
Copyright (c) 2010 __Richard Careaga__ All rights reserved

     Redistribution and use in source and binary forms, with or without
     modification, are permitted provided that the following conditions are
     met:
     * Redistributions of source code must retain the above copyright
       notice, this list of conditions and the following disclaimer.
     * Redistributions in binary form must reproduce the above copyright
       notice, this list of conditions and the following disclaimer in the
       documentation and/or other materials provided with the distribution.
     * Neither the name of the Richard Careaga nor the names of
       contributors may be used to endorse or promote products
       derived from this software without specific prior written permission.

    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDER AND CONTRIBUTORS "AS
    IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
    LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
    FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL RICHARD
    CAREAGA BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
    EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
    PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
    PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY
    OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
    (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
    DAMAGE.
"""
# Obtain various standard helper functions and classes
from __future__ import division        # needs to be first line
import sys
import os
import plac
import urllib2
from collections import defaultdict
from datetime import date
from datetime import datetime
from dateutil.relativedelta import *
from lxml import etree
from StringIO import StringIO
help_message = '''
demonstration: calculate a decrement table for Sequoia 2010-H1 at a constant
prepayment rate assumption modified so that each loan that prepays does so
in full, rather than a curtailment.
Usage: python ./demonstration.py cpr where cpr is a decimal fraction between
0.01 and 1.00, inclusive
For discussion of the code visit http://www.pylaw.org/demonstration.html and to download
http://www.pylaw.org/demonstration.txt
'''
```

```
'''Constants, from Sequoia Mortgage Trust 2010-H1 (http://goo.gl/I9Wi)'''
dealname = 'Sequoia 2010-H1'
bond = 'Class A-1'
replinefile = 'http://www.pylaw.org/dectable.csv'
margin = 2.25       # identical for each loan
index = 0.9410      # assumed constant per 'modeling assumptions'
expfee = 0.2585     # servicing and trustee fees
reset = margin + index - expfee # interest rate calculation on adjustment
                                # dates
pbal = 237838333.0  # initial aggregate principal balance of the loans
obal = 222378000.0  # initial aggregate principal balance of the Class A-1
srpct = obal/pbal   # initial Senior Principal Percentage
cod = date(2010,5,1)# cut-off date
close_month = cod - relativedelta(months=1)
anniversary_month = (cod - relativedelta(months=1)).strftime('%B')
'''stepdown dates'''
stepdown = dict(
stepone = [date(2017,5,1), 1.0],
steptwo = [date(2018,5,1), 0.7],
stepthree = [date(2019,5,1), 0.6],
stepfour = [date(2020,5,1), 0.4],
stepfive = [date(2021,5,1), 0.2]
)
tttdate = date(2013,5,1) # two times test date
num_replines = 16
num_loans = 255
speeds = [0, 0.1, 0.2, 0.3, 0.4, 0.5]

url='http://www.revisedregab.com/xmlsample.xhtml' #XML file of loans

# helper functions
def generateItems(seq):
    for item in seq:
        yield item

def md(lexicon,key, contents):
    """Generic append key, contents to lexicon"""
    lexicon.setdefault(key,[]).append(contents)


class Solver(object):
    '''takes a function, named arg value (opt.) and returns a Solver object
       http://code.activestate.com/recipes/303396/'''
    def __init__(self,f,**args):
        self._f=f
        self._args={}
        # see important note on order of operations in __setattr__ below.
        for arg in f.func_code.co_varnames[0:f.func_code.co_argcount]:
            self._args[arg]=None
        self._setargs(**args)
    def __repr__(self):
        argstring=','.join(['%s=%s' % (arg,str(value)) for (arg,value) in
                            self._args.items()])
```

```
    if argstring:
        return 'Solver(%s,%s)' % (self._f.func_code.co_name, argstring)
    else:
        return 'Solver(%s)' % self._f.func_code.co_name
def __getattr__(self,name):
    '''used to extract function argument values'''
    self._args[name]
    return self._solve_for(name)
def __setattr__(self,name,value):
    '''sets function argument values'''
    # Note - once self._args is created, no new attributes can
    # be added to self.__dict__.  This is a good thing as it throws
    # an exception if you try to assign to an arg which is inappropriate
    # for the function in the solver.
    if self.__dict__.has_key('_args'):
        if name in self._args:
            self._args[name]=value
        else:
            raise KeyError, name
    else:
        object.__setattr__(self,name,value)
def _setargs(self,**args):
    '''sets values of function arguments'''
    for arg in args:
        self._args[arg]  # raise exception if arg not in _args
        setattr(self,arg,args[arg])
def _solve_for(self,arg):
    '''Newton's method solver'''
    TOL=0.0000001      # tolerance
    ITERLIMIT=1000        # iteration limit
    CLOSE_RUNS=10   # after getting close, do more passes
    args=self._args
    if self._args[arg]:
        x0=self._args[arg]
    else:
        x0=1
    if x0==0:
        x1=1
    else:
        x1=x0*1.1
    def f(x):
        '''function to solve'''
        args[arg]=x
        return self._f(**args)
    fx0=f(x0)
    n=0
    while 1:                    # Newton's method loop here
        fx1 = f(x1)
        if fx1==0 or x1==x0:  # managed to nail it exactly
            break
        if abs(fx1-fx0)<TOL:    # very close
            close_flag=True
            if CLOSE_RUNS==0:       # been close several times
                break
```

```
                else:
                    CLOSE_RUNS-=1       # try some more
            else:
                close_flag=False
            if n>ITERLIMIT:
                print "Failed to converge; exceeded iteration limit"
                break
            slope=(fx1-fx0)/(x1-x0)
            if slope==0:
                if close_flag:  # we're close but have zero slope, finish
                    break
                else:
                    print 'Zero slope and not close enough to solution'
                    break
            x2=x0-fx0/slope           # New 'x1'
            fx0 = fx1
            x0=x1
            x1=x2
            n+=1
        self._args[arg]=x1
        return x1

def tvm(pv,fv,pmt,n,i):
    '''equation for time value of money'''
    i=i/100
    tmp=(1+i)**n
    return pv*tmp+pmt/i*(tmp-1)-fv
## end of http://code.activestate.com/recipes/303396/ }}}

class Payoff():
    '''prepares a decrement table given constant prepayment speed'''
    def __init__(self, L, C):
        self.L = L
        self.C = C
        self.bbal = float(L[0])     #beginning balance
        self.rbal = self.bbal       #remaining balance
        self.i = float(L[1])        #interest rate in form 4.5
        self.rtm = int(L[2])        #remaining months to maturity
        self.mtr = int(L[3])+1      #months to roll date new i in effect
        self.mta = int(L[4])        #months remaining of interest only
        self.cod = C[0]             #cut-off date
        self.tttdate = C[1]         #twotimes test date
        self.srpct = C[2]           #initial senior percentage
        self.osrpct = C[2]          #original senior percentage
        self.reset = C[3]           #interest rate at reset
        self.stepdown = C[4]        #stepdown dates
        self.pbal = C[5]            #original aggregate principal balance
        self.obal = C[6]            #original aggregate class balance
        self.obsupct = 1 - C[2]     #original subordinate percentage
        s = Solver(tvm,pv=self.bbal, fv=0, i = self.i/12, n = self.rtm)
        self.pmt = s.pmt            #monthly payment
        self.teaser = self.mtr      #counter for initial fixed rate period
        self.io = self.mta          #counter for remaining interest only
        self.n = self.rtm+1         #to take into account range()
```

```
    self.current = self.cod + relativedelta(months=+1)
    self.smm = 0.0              #single monthly mortality
def __nonzero__(self):
    return True
def __bool__(self):
    return False
def payone(self):
    def is_twice():             #Twotimes test
        if self.subprct >= 2*self.osubpct:
            return 1
        else:
            return 0
    def is_shrinking():
        if self.srpct > self.osrpct:
             return 1
        else:
            return 0
    def payoff():
        import random               #import standard randomization module
        space = int(1//self.smm)    #calculate sample space
        outcomes = [1]              #create list with one positive outcome
        for n in range(space-1):    #for the remainder of the sample space
            outcomes.append(0)      #populate with negative outcome
        payoff = random.choice(outcomes)#randomly choose an outcome
        return payoff                   #report result to calling function
    def senior_prepay_percentage():
        if self.current < self.tttdate and is_twice:
            self.srpppct = self.srpct + 0.5*(1-self.srpct)
        elif self.current >= self.tttdate and is_twice:
            self.srpppct = self.srpct
        elif self.current < self.stepdown['stepone'][0]:
            if is_shrinking():
                self.srpppct = 1.0
            elif is_twice():
                self.srpppct = self.stepdown['stepone'][1]
            else:
                self.srpppct = self.srpppct
        elif self.current < self.stepdown['steptwo'][0]:
            if is_shrinking():
                self.srpppct = 1.0
            elif is_twice():
                self.srpppct = self.stepdown['steptwo'][1]
            else:
                self.srpppct = self.srpppct
        elif self.current < self.stepdown['stepthree'][0]:
            if is_shrinking():
                self.srpppct = 1.0
            elif is_twice():
                self.srpppct = self.stepdown['stepthree'][1]
            else:
                self.srpppct = self.srpppct
        elif self.current < self.stepdown['stepfour'][0]:
            if is_shrinking():
                self.srpppct = 1.0
```

```
        elif is_twice():
            self.srpppct = self.stepdown['stepfour'][1]
        else:
            self.srpppct = self.srpppct
    elif self.current < self.stepdown['stepfive'][0]:
        if is_shrinking():
            self.srpppct = 1.0
        elif is_twice():
            self.srpppct = self.stepdown['stepfive'][1]
        else:
            self.srpppct = self.srpppct
    elif self.current >= self.stepdown['stepfive'][0]:
        self.srpppct = self.srpct
    else:
        self.srpppct = self.srpct
    next_month = self.current + relativedelta(months=+1)
    self.current = next_month
senior_prepay_percentage()  #calculate senior prepayment
                            #percentage
self.teaser -= 1            #reduce remaining teaser period
self.io -= 1                #reduce remaining interest only period
self.bbal = self.rbal       #beginning balance to last period's ending
ipay = self.rbal*self.i/1200    #interest payment portion
if payoff():
    self.smm = 1.0
if self.mta > 0:            #if during interest only period
    self.paydown = 0        #no scheduled principal
    self.prepay = self.smm*(self.bbal-self.paydown)
else:
    self.paydown = -self.pmt-ipay # reverse negative paid out conv
    self.prepay = self.smm*(self.bbal-self.paydown)
if self.rtm > 0:            #decrement remaining term to maturity
    self.rtm -= 1
if self.mtr == 0:           #begin 12-month reset period 11 .. 0
    self.mtr = 11
elif self.mtr > 0:          #decrement months to reset
    self.mtr -= 1
if self.mta > 0:            #decrement months to end of i/o period
    self.mta -= 1
if self.bbal == 0:          #see if final payment has been made
    self.paydown = 0
    self.prepay = 0
elif self.bbal >= self.paydown + self.prepay:  #not last payment?
    self.rbal -= self.paydown + self.prepay
elif self.bbal < self.paydown: # scheduled payment enough to final out
    self.paydown = self.bbal
    self.prepay = 0
    self.rbal = 0
elif self.bbal < self.prepay: # prepayment enough to final out
    self.paydown = self.bbal
    if self.bbal > 0:       # if any still left, allocate to prepay
        self.prepay = self.bbal
        self.rbal = 0
else:
```

```
            self.rbal = 0
        if self.teaser == 1:        #last month of fixed rate period
            self.i = self.reset     #change interest rate for following month
            s = Solver(tvm,pv=self.rbal, fv=0, i = self.i/12, \
            n = self.rtm+1)         #calculate new amortizing payment
            self.pmt = s.pmt        #set new payment
        if self.io == 1:            #last month of i/o period
            s = Solver(tvm,pv=self.rbal, fv=0, i = self.i/12, \
            n = self.rtm)           #calculate amortizing payment
            self.pmt = s.pmt        #set new payment
        yield self.srpct*self.paydown + self.srpppct*self.prepay

#create an empty dictionary for each loan record
d1 = defaultdict(list)
d2 = defaultdict(list)
d3 = defaultdict(list)
d4 = defaultdict(list)
d5 = defaultdict(list)
d6 = defaultdict(list)
d7 = defaultdict(list)
d8 = defaultdict(list)
d9 = defaultdict(list)
d10 = defaultdict(list)
d11 = defaultdict(list)
d12 = defaultdict(list)
d13 = defaultdict(list)
d14 = defaultdict(list)
d15 = defaultdict(list)
d16 = defaultdict(list)
d17 = defaultdict(list)
d18 = defaultdict(list)
d19 = defaultdict(list)
d20 = defaultdict(list)
d21 = defaultdict(list)
d22 = defaultdict(list)
d23 = defaultdict(list)
d24 = defaultdict(list)
d25 = defaultdict(list)
d26 = defaultdict(list)
d27 = defaultdict(list)
d28 = defaultdict(list)
d29 = defaultdict(list)
d30 = defaultdict(list)
d31 = defaultdict(list)
d32 = defaultdict(list)
d33 = defaultdict(list)
d34 = defaultdict(list)
d35 = defaultdict(list)
d36 = defaultdict(list)
d37 = defaultdict(list)
d38 = defaultdict(list)
d39 = defaultdict(list)
d40 = defaultdict(list)
d41 = defaultdict(list)
```

```
d42 = defaultdict(list)
d43 = defaultdict(list)
d44 = defaultdict(list)
d45 = defaultdict(list)
d46 = defaultdict(list)
d47 = defaultdict(list)
d48 = defaultdict(list)
d49 = defaultdict(list)
d50 = defaultdict(list)
d51 = defaultdict(list)
d52 = defaultdict(list)
d53 = defaultdict(list)
d54 = defaultdict(list)
d55 = defaultdict(list)
d56 = defaultdict(list)
d57 = defaultdict(list)
d58 = defaultdict(list)
d59 = defaultdict(list)
d60 = defaultdict(list)
d61 = defaultdict(list)
d62 = defaultdict(list)
d63 = defaultdict(list)
d64 = defaultdict(list)
d65 = defaultdict(list)
d66 = defaultdict(list)
d67 = defaultdict(list)
d68 = defaultdict(list)
d69 = defaultdict(list)
d70 = defaultdict(list)
d71 = defaultdict(list)
d72 = defaultdict(list)
d73 = defaultdict(list)
d74 = defaultdict(list)
d75 = defaultdict(list)
d76 = defaultdict(list)
d77 = defaultdict(list)
d78 = defaultdict(list)
d79 = defaultdict(list)
d80 = defaultdict(list)
d81 = defaultdict(list)
d82 = defaultdict(list)
d83 = defaultdict(list)
d84 = defaultdict(list)
d85 = defaultdict(list)
d86 = defaultdict(list)
d87 = defaultdict(list)
d88 = defaultdict(list)
d89 = defaultdict(list)
d90 = defaultdict(list)
d91 = defaultdict(list)
d92 = defaultdict(list)
d93 = defaultdict(list)
d94 = defaultdict(list)
d95 = defaultdict(list)
```

```
d96 = defaultdict(list)
d97 = defaultdict(list)
d98 = defaultdict(list)
d99 = defaultdict(list)
d100 = defaultdict(list)
d101 = defaultdict(list)
d102 = defaultdict(list)
d103 = defaultdict(list)
d104 = defaultdict(list)
d105 = defaultdict(list)
d106 = defaultdict(list)
d107 = defaultdict(list)
d108 = defaultdict(list)
d109 = defaultdict(list)
d110 = defaultdict(list)
d111 = defaultdict(list)
d112 = defaultdict(list)
d113 = defaultdict(list)
d114 = defaultdict(list)
d115 = defaultdict(list)
d116 = defaultdict(list)
d117 = defaultdict(list)
d118 = defaultdict(list)
d119 = defaultdict(list)
d120 = defaultdict(list)
d121 = defaultdict(list)
d122 = defaultdict(list)
d123 = defaultdict(list)
d124 = defaultdict(list)
d125 = defaultdict(list)
d126 = defaultdict(list)
d127 = defaultdict(list)
d128 = defaultdict(list)
d129 = defaultdict(list)
d130 = defaultdict(list)
d131 = defaultdict(list)
d132 = defaultdict(list)
d133 = defaultdict(list)
d134 = defaultdict(list)
d135 = defaultdict(list)
d136 = defaultdict(list)
d137 = defaultdict(list)
d138 = defaultdict(list)
d139 = defaultdict(list)
d140 = defaultdict(list)
d141 = defaultdict(list)
d142 = defaultdict(list)
d143 = defaultdict(list)
d144 = defaultdict(list)
d145 = defaultdict(list)
d146 = defaultdict(list)
d147 = defaultdict(list)
d148 = defaultdict(list)
d149 = defaultdict(list)
```

```
d150 = defaultdict(list)
d151 = defaultdict(list)
d152 = defaultdict(list)
d153 = defaultdict(list)
d154 = defaultdict(list)
d155 = defaultdict(list)
d156 = defaultdict(list)
d157 = defaultdict(list)
d158 = defaultdict(list)
d159 = defaultdict(list)
d160 = defaultdict(list)
d161 = defaultdict(list)
d162 = defaultdict(list)
d163 = defaultdict(list)
d164 = defaultdict(list)
d165 = defaultdict(list)
d166 = defaultdict(list)
d167 = defaultdict(list)
d168 = defaultdict(list)
d169 = defaultdict(list)
d170 = defaultdict(list)
d171 = defaultdict(list)
d172 = defaultdict(list)
d173 = defaultdict(list)
d174 = defaultdict(list)
d175 = defaultdict(list)
d176 = defaultdict(list)
d177 = defaultdict(list)
d178 = defaultdict(list)
d179 = defaultdict(list)
d180 = defaultdict(list)
d181 = defaultdict(list)
d182 = defaultdict(list)
d183 = defaultdict(list)
d184 = defaultdict(list)
d185 = defaultdict(list)
d186 = defaultdict(list)
d187 = defaultdict(list)
d188 = defaultdict(list)
d189 = defaultdict(list)
d190 = defaultdict(list)
d191 = defaultdict(list)
d192 = defaultdict(list)
d193 = defaultdict(list)
d194 = defaultdict(list)
d195 = defaultdict(list)
d196 = defaultdict(list)
d197 = defaultdict(list)
d198 = defaultdict(list)
d199 = defaultdict(list)
d200 = defaultdict(list)
d201 = defaultdict(list)
d202 = defaultdict(list)
d203 = defaultdict(list)
```

```
d204 = defaultdict(list)
d205 = defaultdict(list)
d206 = defaultdict(list)
d207 = defaultdict(list)
d208 = defaultdict(list)
d209 = defaultdict(list)
d210 = defaultdict(list)
d211 = defaultdict(list)
d212 = defaultdict(list)
d213 = defaultdict(list)
d214 = defaultdict(list)
d215 = defaultdict(list)
d216 = defaultdict(list)
d217 = defaultdict(list)
d218 = defaultdict(list)
d219 = defaultdict(list)
d220 = defaultdict(list)
d221 = defaultdict(list)
d222 = defaultdict(list)
d223 = defaultdict(list)
d224 = defaultdict(list)
d225 = defaultdict(list)
d226 = defaultdict(list)
d227 = defaultdict(list)
d228 = defaultdict(list)
d229 = defaultdict(list)
d230 = defaultdict(list)
d231 = defaultdict(list)
d232 = defaultdict(list)
d233 = defaultdict(list)
d234 = defaultdict(list)
d235 = defaultdict(list)
d236 = defaultdict(list)
d237 = defaultdict(list)
d238 = defaultdict(list)
d239 = defaultdict(list)
d240 = defaultdict(list)
d241 = defaultdict(list)
d242 = defaultdict(list)
d243 = defaultdict(list)
d244 = defaultdict(list)
d245 = defaultdict(list)
d246 = defaultdict(list)
d247 = defaultdict(list)
d248 = defaultdict(list)
d249 = defaultdict(list)
d250 = defaultdict(list)
d251 = defaultdict(list)
d252 = defaultdict(list)
d253 = defaultdict(list)
d254 = defaultdict(list)
d255 = defaultdict(list)
websters = [d1, d2, d3, d4, d5, d6, d7, d8, d9, d10, d11, d12, d13, d14, d15, d16, d17, d18,
d19, d20, d21, d22, d23, d24, d25, d26, d27, d28, d29, d30, d31, d32, d33, d34, d35, d36,
```

```
d37, d38, d39, d40, d41, d42, d43, d44, d45, d46, d47, d48, d49, d50, d51, d52, d53, d54,
d55, d56, d57, d58, d59, d60, d61, d62, d63, d64, d65, d66, d67, d68, d69, d70, d71, d72,
d73, d74, d75, d76, d77, d78, d79, d80, d81, d82, d83, d84, d85, d86, d87, d88, d89, d90,
d91, d92, d93, d94, d95, d96, d97, d98, d99, d100, d101, d102, d103, d104, d105, d106, d107,
d108, d109, d110, d111, d112, d113, d114, d115, d116, d117, d118, d119, d120, d121, d122,
d123, d124, d125, d126, d127, d128, d129, d130, d131, d132, d133, d134, d135, d136, d137,
d138, d139, d140, d141, d142, d143, d144, d145, d146, d147, d148, d149, d150, d151, d152,
d153, d154, d155, d156, d157, d158, d159, d160, d161, d162, d163, d164, d165, d166, d167,
d168, d169, d170, d171, d172, d173, d174, d175, d176, d177, d178, d179, d180, d181, d182,
d183, d184, d185, d186, d187, d188, d189, d190, d191, d192, d193, d194, d195, d196, d197,
d198, d199, d200, d201, d202, d203, d204, d205, d206, d207, d208, d209, d210, d211, d212,
d213, d214, d215, d216, d217, d218, d219, d220, d221, d222, d223, d224, d225, d226, d227,
d228, d229, d230, d231, d232, d233, d234, d235, d236, d237, d238, d239, d240, d241, d242,
d243, d244, d245, d246, d247, d248, d249, d250, d251, d252, d253, d254, d255]

content = urllib2.urlopen(url).read()
root = etree.fromstring(content)
records = list(root)
lexicon = generateItems(websters)
for record in records:
    lex = lexicon.next()
    for field in record:
       md(lex, field.attrib['name'], field.text)

tape = []
for loan in websters:
    record = []
    record.append(float(loan['obal'][0]))
    record.append(float(loan['cintpct'][0]))
    tmat = loan['maturity'][0]
    mat = datetime.strptime(tmat, '%Y-%m-%d').date()
    to_mat = relativedelta(mat,cod)
    mtm = to_mat.months + to_mat.years*12
    record.append(mtm)
    fpd = datetime.strptime(loan['fpd'][0], '%Y-%m-%d').date()
    to_roll = relativedelta(fpd + relativedelta(months=60), cod)
    mtr = to_roll.months + to_roll.years*12
    record.append(mtr)
    intonlyterm = int(loan['intonlyterm'][0])
    to_amort = relativedelta(fpd + relativedelta(months=intonlyterm), cod)
    mta = to_amort.months + to_amort.years*12
    record.append(mta)
    tape.append(record)

def run_loan_payoff(cpr):
    '''cpr = 0.1 Constant Prepayment Rate in decimal fraction'''
    C = [cod, tttdate, srpct, reset, stepdown, pbal, obal]
    cbal = obal
    anniversary = cod.year+1
    E = {}
    for record in tape:
        md(E,'tape', Payoff(record,C))
    twelfth = 1.0/12.0
    smm = 1.0 - (1.0-cpr)**twelfth  #  single monthly mortality
    column = []                     # empty list to collect principal payments
```

```
    for year in range(2011,2041):
        annual = []                # temporary list
        for month in range(12):
            for entry in E['tape']:
                payment = []       # temporary list
                entry.srpct = srpct # set object senior percentage
                entry.subpct = 1 - srpct
                entry.smm = smm     # set smm for object
                try:               # while still data
                    payment.append(entry.payone().next())
                except StopIteration:
                    pass
                annual.append(sum(payment))    # aggregate for month
                cbal -= sum(payment)           # knock down senior
            sprct = cbal/obal                  # recalculate senior percentage
        column.append(annual)                  # collect months
    column[:] = [sum(item) for item in column] # aggregate for year
    cbal=obal
    ''' output decrement table for given CPR speed '''
    print "%s %s at CPR of %d%%" % (dealname, bond, cpr*100)
    for year in column:
        cbal -= year
        percentout = round(cbal/obal*100,2)
        if percentout >= 1:
            print("%s %d:\t\t%0.0f") % (anniversary_month,  anniversary,\
            percentout)
        elif percentout <= 0:
            print("%s %d:\t\t0") % (anniversary_month, anniversary)
        else:
            percentout < 1
            print("%s %d:\t\t*") % (anniversary_month, anniversary)
        anniversary += 1

def main(cpr_arg):
    print help_message
    cpr = float(cpr_arg) # command line argument is a string
    run_loan_payoff(cpr) # call the function to produce the table

if __name__ == "__main__":
        plac.call(main)
```