```
# File komega-v05a.py

# Author: G.Doeben-Henisch
# First date: September 4, 2020
# Last change: 13.September 2020

##################################
# Execution Environment of my local machine:
# (venv) gerd@gerd-ub2:~/env/komega/tst$ python3 komega-v01d.py
#

####################################
# GITHUB
#
# We use a github repository at:
# https://github.com/szmt/komega.git
#
# Im working from a unix-shell using the following github-commands:
# https://git-scm.com/docs/git

####################################
# BACKGROUND THEORY
#
# This code is a translation of a theory described in the blog
# https://www.uffmm.org
#
# Last document for the specification of this code:
#
'''
https://www.uffmm.org/2020/09/10/komega-requirements-no-4-version-4-basic-application-scenario/
'''
############################################
# HMI - CLASS PUBLISH
#
'''
The intended interaction of the user with the system will be realized through an interactive web page. In this experimental program there is no web page but a normal console. Therefore (proposal from Tobias Schmitt) we have a special class 'Publish' which handles all console input and output and the other classes interact with this class Publish. In the context of the web server we can then replace the class Publish by appropriate libraries for HTML web pages.
'''
####################################
# ACTOR STORY
#
# In the specifications an actor story [AS] has been specified. This AS requires # some basic states which are dedicated for certain tasks to do:

'''
ACTOR STORY

S1: START
```

S2: EDIT P(roblem description)
S3: EDIT S (actual state)
S4: EDIT X (change rules)
S5: SIMULATION (Applying X to S)
S6: EVALUATION (After the simulation)
S7: STOP
'''

MAIN IDEA
'''
According to the above mentioned actor story the user will be sitting in front of a system interface [SI] which works first only as a console.
In the beginning the user is placed in a start state S1 showing all options available.
The user can select one of these options and can from start state S1 reach all other states S2-S7.
'''

##############################
IMPORTS

##############################
SUPPORTING FUNCTION
#

No funtions yet

CLASSES
#
'''
For every state there exists one working class to do the job.
The special class 'Publish' in this code exists only because the interaction of the user with the system will happen with an interactive website which uses HTML and javascript. Here in this experimental environment a simple unix-console is used.
'''

import kcv5a as kc

##########################
Main Programm
#

###############
Start main loop
#
The loop will work as long as the value of the variable 'loop' is different to 'N'

loop='Y'
while loop!='N':

################################
STATE 1 : START
Show available options

```
# Get feedback for selection
# Confirm the selection
# Distribute to different states

        kc.ast.menushow()

# Ask back for selection number
        message='Enter a Number [1-7] for Menu Option \n'
        kc.pub.userinput(message)

# Evaluate the selection

        opt=kc.pub.opt
        kc.ast.badoption(opt)

#####################################################################
# Call to a class instance
#
#####################################################################
# Call to state Edit Problem P
################################
# STATE 2 : EDIT P
# Ask Questions related to P
# Collect all answers into one problem document
#

        if opt=='2':
                # Where You are
                kc.pub.show(kc.ap)

                #Interaction with Problem Class
                message='Enter your problem as it is now given in plain text\n'
                kc.pub.userinput(message)
                inp=kc.pub.opt
                kc.app.getproblem(inp)

                message='Enter your vision of a better state in the future in plain text\n'
                kc.pub.userinput(message)
                inp=kc.pub.opt
                kc.app.getvision(inp)

                message='Enter the name of the city you are in\n'
                kc.pub.userinput(message)
                inp=kc.pub.opt
                kc.app.getregion(inp)

                message='Time model [From, Until,Cycleunit [Y or M or D or H]]: '
                kc.pub.userinput(message)
                inp=kc.pub.opt
                kc.app.gettime(inp)
```

```
                message='Which kinds of persons are important? Write a list, comma separated
please : '
                kc.pub.userinput(message)
                inp=kc.pub.opt
                kc.app.getperson(inp)

                kc.app.problemTotal()

                ########################################
                # Put the information from problem into the document 'docp'

                message='Your Problem document is now :\n'
                kc.pub.useroutput(message)
                docp=kc.app.problemAll
                kc.pub.useroutput(docp)



#
#####################################################################
# Call to state Edit Actual State S
################################
# STATE 3 : EDIT S
# Collect single expressions
# Collect all expressions into one  document describing S
#

        elif opt=='3':
                # Where You are
                kc.pub.show(kc.ass)

                Sloop='Y'
                while Sloop!='N':

                        # Interaction with actual state S class
                   message='Enter an expression for your state description  in plain text\n'
                   kc.pub.userinput(message)
                   inp=kc.pub.opt
                   kc.aas.getexpression(inp)
                   message="STOP Editing S = 'N', CONTINUE != 'N'  \n"
                   kc.pub.userinput(message)
                   inp=kc.pub.opt
                   Sloop=inp

                ######################################################
                # Keeping the document

                message='Your State Description document is now :\n'
                kc.pub.useroutput(message)
                docs=kc.aas.stateAll
                kc.pub.useroutput(docs)
```

```
#
#####################################################################
# Call to state Edit Change Rules X
################################
# STATE 4 : EDIT X
# Collect single expressions for change rules
# Collect all expressions into one  document describing X
#

        elif opt=='4':
                # Where You are
                kc.pub.show(kc.ax)

                Xloop='Y'
                while Xloop!='N':

                        kc.axx.rule=[]

                        # Interaction with actual X class
                        message="Enter a rule for your xchange rules in plain text with the parts
CONDITION, PROBABILITY, EFFEKT-, EFFEKT+ \n"
                        kc.pub.useroutput(message)
                        message="We will ask You for each category separatedly :\n"
                        kc.pub.useroutput(message)

                        message=kc.axx.rcat[0]+' : '
                        kc.pub.userinput(message)
                        inp=kc.pub.opt
                        kc.axx.getrule(inp)

                        message=kc.axx.rcat[1]+' : '
                        kc.pub.userinput(message)
                        inp=kc.pub.opt
                        kc.axx.getrule(inp)

                        message=kc.axx.rcat[2]+' : '
                        kc.pub.userinput(message)
                        inp=kc.pub.opt
                        kc.axx.getrule(inp)

                        message=kc.axx.rcat[3]+' : '
                        kc.pub.userinput(message)
                        inp=kc.pub.opt
                        kc.axx.getrule(inp)

                        kc.axx.rulesAll.append(kc.axx.rule)
                        kc.axx.rule=[]
                        kc.axx.rulessummary()

                        message="STOP Editing X = 'N', CONTINUE != 'N'  \n"
```

```
                kc.pub.userinput(message)
                inp=kc.pub.opt
                Xloop=inp

            ##########################################################
            # Keeping the document

            message='Your Rules  document is now :\n'
            kc.pub.useroutput(message)
            docx=kc.axx.rulesAll
            kc.pub.useroutput(docx)

#
#########################################################################
# Call to state  Simulation SIM
################################
# STATE 5 : Run the simulation
#


        elif opt=='5':
            kc.pub.show(kc.asim)

#
#########################################################################
# Call to state  Evaluation EV
#

        elif opt=='6':
            kc.pub.show(kc.aev)

#
#########################################################################
# Call to state Stop STP
#

        elif opt=='7':
            kc.pub.show(kc.astp)

#################################################################
# End of Loop

# Clarify how to continue

        message="STOP MAIN LOOP = 'N', CONTINUE != 'N'  \n"
        kc.pub.userinput(message)
        inp=kc.pub.opt
        loop=inp

'''
```

TEST NEW CLASS X

```
(venv) gerd@gerd-ub2:~/env/komega/tst$ python3 komega-v05a.py
1 is START
2 is EDIT P
3 is EDIT S
4 is EDIT X
5 is SIMULATION
6 is EVALUATION
7 is STOP
Enter a Number [1-7] for Menu Option
4
!!You have selected the state :
EDIT X
Here You can edit some change rules X to apply to an actual state S.
Enter a rule for your xchange rules in plain text with the parts CONDITION, PROBABILITY,
EFFEKT-, EFFEKT+

We will ask You for each category separatedly :

CONDITION : Mary needs a book
Your single rule buffer : []
Feedback Your last input :
Mary needs a book
Your single rule buffer : ['Mary needs a book']
PROBABILITY : 0.9
Your single rule buffer : ['Mary needs a book']
Feedback Your last input :
0.9
Your single rule buffer : ['Mary needs a book', '0.9']
EFFECT- : non
Your single rule buffer : ['Mary needs a book', '0.9']
Feedback Your last input :
non
Your single rule buffer : ['Mary needs a book', '0.9', 'non']
EFFECT+ : Mary enters library
Your single rule buffer : ['Mary needs a book', '0.9', 'non']
Feedback Your last input :
Mary enters library
Your single rule buffer : ['Mary needs a book', '0.9', 'non', 'Mary enters library']
Feedback Your rules  so far :
[['Mary needs a book', '0.9', 'non', 'Mary enters library']]
STOP Editing X = 'N', CONTINUE != 'N'
a
Enter a rule for your xchange rules in plain text with the parts CONDITION, PROBABILITY,
EFFEKT-, EFFEKT+

We will ask You for each category separatedly :

CONDITION : Mary enters library
Your single rule buffer : []
Feedback Your last input :
```

```
Mary enters library
Your single rule buffer : ['Mary enters library']
PROBABILITY : 0.1
Your single rule buffer : ['Mary enters library']
Feedback Your last input :
0.1
Your single rule buffer : ['Mary enters library', '1.0']
EFFECT- : Mary enters library
Your single rule buffer : ['Mary enters library', '1.0]
Feedback Your last input :
Mary enters library
Your single rule buffer : ['Mary enters library', '1.0', 'Mary enters library']
EFFECT+ : Mary is in the library
Your single rule buffer : ['Mary enters library', '1.0', 'Mary enters library']
Feedback Your last input :
Mary is in the library
Your single rule buffer : ['Mary enters library', '1.0', 'Mary enters library', 'Mary is in the library']
Feedback Your rules  so far :
[['Mary needs a book', '0.9', 'non', 'Mary enters library'], ['Mary enters library', '1.0', 'Mary enters
library', 'Mary is in the library']]
STOP Editing X = 'N', CONTINUE != 'N'
N
Your Rules  document is now :

[['Mary needs a book', '0.9', 'non', 'Mary enters library'], ['Mary enters library', '1.0', 'Mary enters
library', 'Mary is in the library']]
STOP MAIN LOOP = 'N', CONTINUE != 'N'
N
(venv) gerd@gerd-ub2:~/env/komega/tst$
'''


 # File kcv5a.py

# Author: G.Doeben-Henisch
# First date: September 6, 2020
# Last date: September  13, 2020

#################################
# CLASS DEFINITIONS

class Start:
        def __init__(self):
                self.menulist = ['START','EDIT P','EDIT S', 'EDIT
X','SIMULATION','EVALUATION','STOP']

        def menushow(self):
                i=0  # Counter for menu-loop
                for state in self.menulist:
                        i=i+1
                        message="str(i)+' is '+state"
                        pub.useroutput(eval(message))
```

```
        def badoption(self,opt):
                if int(opt)<1 or int(opt)>7:
                        message='!!You have selected a bad option'
                        pub.useroutput(message)

                if int(opt)>0 and int(opt)<8:
                        message='!!You have selected the state :\n'+self.menulist[int(opt)-1]
                        pub.useroutput(message)

#####################################################################

class Actor:
        def __init__(self,inp):
                self.message=inp

#####################################################################

class Publish():

        def show(self,other):
                print(other.message)

        def useroutput(self,message):
                print(message)

        def userinput(self,message):
                self.opt=input(message)




#####################################################################
# CLASS PROBLEM
'''
MAIN IDEA

A main window W1 with a menu showing all possible questions to be
answered.

(a) Describe the problem P: What is given and what is the intended future state?
(b) Describe the intended real part of the world (space).
(c) Describe the time model T : which time period, which cycles.
(d) Which kinds of actors are seen as being important for the problem and its  future?
(e) Some other assumptions.
'''


class Problem(Actor):

        def getproblem(self,inp):
                self.problemNow = inp
```

```
            message='Feedback Problem Now :\n'+self.problemNow
            pub.useroutput(message)

    def getvision(self,inp):
            self.problemFuture = inp
            message='Feedback Problem Future :\n'+self.problemFuture
            pub.useroutput(message)

    def getregion(self,inp):
            self.problemRegion = inp
            message='Feedback Problem Region :\n'+self.problemRegion
            pub.useroutput(message)

    def gettime(self,inp):
            self.problemTime = inp
            self.problemTM = self.problemTime.split(',')
            message='Feedback Problem TimeModel :\n'+str(self.problemTM)
            pub.useroutput(message)

    def getperson(self,inp):
            self.problemPerson = inp
            self.problemPRS = self.problemPerson.split(',')
            message='Feedback Problem Persons :\n'+str(self.problemPRS)
            pub.useroutput(message)

    def problemTotal(self):
            self.problemAll =[]
            self.problemAll.append(self.problemNow)
            self.problemAll.append(self.problemFuture)
            self.problemAll.append(self.problemRegion)
            self.problemAll.append(self.problemTime)
            self.problemAll.append(self.problemPerson)
            message='Feedback Problem All :\n'+str(self.problemAll)
            pub.useroutput(message)

######################################################################
# CLASS S(tate Description)
'''
IDEA:
This state should allow in the final version the editing of the texts S and X in parallel. Additionally
one should be able to call from within this state(s) the simulation mode to test whether the actual
texts are working.

FOR NOW:
In this first experimental version one has to work either with the stae S or with the state X
separatedly. Simulation would be a follow up state.

TASK:
Input all data which are necessary for the S-state (including sectioning
and extended texts with details)

ACTORS:
```

Human experts.

SYSTEM INTERFACE:
A main window W1 offering the editing of a text consisting of individual statements. Every statement can be edited separately and repeatedly.

ACTIONS:
Select either a given statement for editing or edit a new statement or stop.

IMPLEMENTATION:
Using the list-construct of python to collect expressions, because lists are ordered and mutable and allow many intersting operations.
'''

```
class AState(Actor):

        def __init__(self):
                self.stateAll = []

        def getexpression(self,inp):
                self.expression = inp
                self.stateAll.append(inp)
                message='Feedback Your last expression :\n'+str(self.expression)
                pub.useroutput(message)
                message='Feedback Your document S so far :\n'+str(self.stateAll)
                pub.useroutput(message)
```

```
#############################################################################
# CLASS X (Change Rules)
'''
```

IDEA:
The change rules X are described in the requirements paper cited in the beginning of the main program text. The principal idea of the change rules X is to allow changes to an actual state S if certain conditions are fulfilled (satisfied). These changes will be executed during the state called simulation.

FOR NOW:
Because a complete implementation of the theoretically possible change rules is nearly an infinite task this version of the change rules X called X01 is limited to the simplest possible case. This contains the following simple structure:

IF Condition C THEN with Probability Pr realize the Effect E- and E+.

While this is already the case without any actor all the parts (C,E+,E+) are additionally limited to one expression each. Thus we start with the format:

IF Condition C(1) THEN with Probability Pr [0,1] realize the Effect E-(1) and E+(1).

The strategy is to extend all these limits stepwise in the next versions.

TASK:
Input all rules for the X-state

ACTORS:
Human experts.

SYSTEM INTERFACE:
A main window W1 offering the editing of a text consisting of individual
rules. Every statement can be edited separately and repeatedly.

ACTIONS:
Select either a given statement for editing or edit a new statement or
stop.

IMPLEMENTATION:
Using the list-construct of python to collect expressions, because lists are ordered and mutable and allow many intersting operations.
'''

```
class Xrules(Actor):

        def __init__(self):
                self.rulesAll = []
                self.rule=[]
                self.rcat=['CONDITION', 'PROBABILITY','EFFECT-', 'EFFECT+']

        def greeting(self):
                message="Your rule set at start :"+str(self.rulesAll)
                pub.useroutput(message)
                message="Your single rule buffer at start :"+str(self.rule)
                pub.useroutput(message)

        def getrule(self,inp):
                message="Your single rule buffer : "+str(self.rule)
                pub.useroutput(message)
                self.rule.append(inp)
                message='Feedback Your last input :\n'+inp
                pub.useroutput(message)
                message="Your single rule buffer : "+str(self.rule)
                pub.useroutput(message)

        def rulessummary(self):
                message='Feedback Your rules  so far :\n'+str(self.rulesAll)
                pub.useroutput(message)



######################################################################
# CLASS INSTANCES

ast=Start()

ap=Actor("Here you can describe your problem with regard to different questions.")
```

app=Problem("Here you can describe your problem with regard to different questions.")

ass=Actor("Here You can describe an actual state S related to your problem.")
aas=AState()
ax=Actor("Here You can edit some change rules X to apply to an actual state S.")
axx=Xrules()
asim=Actor("Here You can run a simulation SIM to check what happens with your initial state S when the change rules X will be applied repeatadly on the state S.")
aev=Actor("Here some advice will be given how to organize an evaluation EVAL of a realized simulation SIM.")
astp=Actor("This will stop the whole program.")

pub=Publish()