The Cost of a Tourist Tour – a Database of Knowledge Written in Python

S. Sveleba, I. Kunyo, I. Karpa
Ivan Franko National University of Lviv
107 Tarnavsky St.,
UA–79017 Lviv, Ukraine
incomlviv@gmail.com

N. Sveleba
Lviv Institute of Economics and Tourism
8 Mentsynskyi St.,
UA–79007 Lviv, Ukraine
incomlviv@gmail.com

Abstract—**The Python language demo database is created in the work, which contains the concept of the value of the tour and the factors that affect this value and the causal relationship between these concepts.**

Index Terms—**knowledge base; Python language; people recreation.**

By definition, the knowledge base is a set of facts and rules of logical conclusion (output) in the chosen subject area. Rules of logical conclusion are the rules for transforming the original system of facts (judgments) into a new system of facts (conclusions). A program that executes a logical conclusion from a pre-built base of facts and rules in accordance with the laws of formal logic, is called the output machine. The knowledge base and the output machine are the main components of the expert system - a program that is designed to find ways to solve problems in a specific subject area. [1, 2].

At present, a lot of knowledge has been accumulated concerning the definition of the cost of services of tourist enterprises. However, there is a problem of the effective use of this knowledge by people who are planning recreation. Literature and other sources of expert information on cost (price) and quality of tourist services mainly contain a set of facts in the form of causal relationships and relationships between factors that affect the cost of travel services. There is a problem of efficient presentation of such knowledge. The main idea is to use as a knowledge representation language and to create queries to the knowledge base of the Python programming language [4, 5], which, unlike the special languages, is a general purpose language and therefore gives the developer of the expert system much more opportunities. To do this, you can use object-oriented capabilities of Python and its introspection, since it is known that such methods of presentation of clear knowledge, such as frames and semantic networks, can be easily implemented by means of object-oriented programming.

Python supports full runtime introspection. That is, for any object during execution it is possible to obtain all information about its structure [4]. For example, the most well-known introspection tool in Python is the dir () function, which returns a list of attribute names for the object passed to it. The function type () or the __class__ attribute allows you to get the type of object. The vars () function or the __dict__ attribute allows you to get the dictionary with the pairs attribute: the value of the object. The functions hasattr (), getattr () and setattr () allow you to check the presence of an attribute in the object, return it and change the value. The issubclass () function allows you to determine whether one class is inherited from another, and the __subclasses __ () method returns the subclass list. The parenthesis tuple and their hierarchy can be obtained using the __bases__ and __mro__ attributes. The inspect module contains additional features that help you get information about objects at runtime. The use of introspection allows you to effectively program output mechanisms and queries to the knowledge base.

The Python language demo database is created in the work, which contains the concept of the value of the tour and the factors that affect this value and the causal relationship between these concepts. To construct a knowledge base, a framing model of knowledge representation has been applied. The Python module, which contains the knowledge base and queries for it, must have a block structure - classes are created first, then individuals, then the properties of individuals are specified, and in the end, requests are executed. Ontology classes correspond to the Python language classes, and subclasses can be created using the Python inheritance mechanism. Thus, the base class Base (describes all objects that have a name) and the classes that inherit it are developed: Factor (describes the factor that affects the cost of the tour), Fact (describes the fact in the form of a triplet subject-predicate- object), Reference (describes the reference to the source of the fact), Dependence (describes the dependence of the value of X on the value of Y).

Individual ontology matches Python objects - the value of the KB dictionary. This method allows you to address individuals by their name in the form of a Unicode string. Individuals can have properties in the form of Python attributes, which allow describing the relationship between individuals. Properties are objects of the Property class, have attributes subj (the property of the property is an individual that has this property), name (property name), inverseName (inverse property name), functional (defines whether the property is functional), symmetric (defines whether the property is symmetric), transitive (determines whether the property is transitive), set (set of power values), and methods __init __ () (constructor, creates the property), add () (adds an object or tuple of objects to a plurality) and __call __ () (returns a set of properties values). The last two methods implement the elements of the output machine. Such items may also be present in queries to the knowledge base.

Knowledge base contains individuals "fatigue", "corrosion", "stress concentration", "cyclic load", "aggressive environment", etc. - objects of the corresponding classes. The created individual object is automatically added to the KB dictionary. Block creation of objects of individuals looks like this:
KB = {} # Knowledge Base Dictionary

Factor ('price') # create object-factor 'price'; Factor ('tour price'); Factor ('hotel'); Factor ('number standard'); Factor ('semi-suite'); Factor ('deluxe'); Factor ('hotel type'); Factor ('transport services'); Factor ('nutrition'); Factor ('excursions'); Factor ('parking'); Factor ('coastline'); Factor ('tour duration'); Factor ('seasonality') ; Factor ('wi-fi'); Factor ('banking services'); Factor ('pool'); Factor ('SPA'); Factor ('playgrounds') Reference ('Vacation in Spain [Electronic resource] - Access mode: http://travel-world.pp.ua/'); Reference ('All hotels in the world [Electronic resource] - Access mode: http://bestdealsonhotel.com/'); Reference ('Hotels in Spain [Electronic resource] - Access mode: http://costagarant.com/'); Dependence (name = 'Dependence of the price of a standard number from time of year', xy = [(0.6.575), (0.7.620), (0.8.630), (0.9.600), (0.10.550)], relative = 'is the minimum') # block creation of facts; # subject 'tour price' 'is the reason' of the object 'lasting tour'; Fact ('tour price', 'isEffect', 'duration of the tour'); Fact ('tour price', 'isEffect', 'hotel type'); Fact ('tour price', 'isEffect', 'seasonality')

The Base class (cost of the tour) contains the attribute name (object name) and the properties And, Or, Not, which allow you to create new objects using logical operations. The Factor class also contains the isCause (is the cause) and isEffect (is a consequence) property. The Fact class contains attributes subjName (subject name), propName (name of the predicate - properties), objName (object name) and hasReference properties (has a link), hasDependence (has a dependency). The Reference class contains the isReference property (a link). The Dependence class contains attributes xy (relative data), relative attribute, isDependence property, and plot function (depicts a dependency graph) and interp (finds the value of linear interpolation). The property is created when creating an individual by calling its constructor __init __ () with parameters subj, name, inverseName, functional, symmetric, transitive, which describe its characteristics. So the isCause and isEffect properties are mutually inverse. This allows, for example, from the statement "the price of a tour is a consequence of the duration of the tour" to make a logical conclusion "the duration of the tour determines the price of the tour." They are also transitive. Properties can be functional, that is, have a unique value, and are symmetric, for example, with the statement "X is Y" to make a logical conclusion "Y is X". Values in the property are added using the add () method.

The block of adding values to the properties of individuals looks like this:

# 'tour price' 'is the reason for' hotel type '

KB ['tour price'].isCause.add (KB ['hotel type']) KB ['tour price'].isCause.add (KB ['power']) KB ['tour price'].isEffect.add (KB ['trips']) KB ['tour duration'].isCause.add (KB ['trips'], KB ['coastline']) KB ['tour price'].and.add (KB ['parkawka'], KB ['SPA']) KB ['tour price'].hasReference.add (KB ['Book your stay in Seville and get exclusive discounts on the city's main attractions [Electronic resource] - Access mode: https://www.booking.com/city / es / sevilla.uk.html '])

KB [price of the tour.isCause.sensibility '].hasDependence.add (KB [' Standard Room Price Dependency from the Seasons'])

Requests for knowledge base are created by searching for objects in sets. You can use Python for and if statements to do this. To access the sets, use the KB object, the property class __call __ (), and standard math operations over sets. The method __call __ () with the showTransitive = True parameter returns the set of all values of the transitive property. For example, a query that displays all the reasons for the "price" factor:
for x in KB ['price'].isEffect (True):

print x.name + '|',

The following query shows facts associated with a particular source:
for x in KB ['Vacation in Spain [Electronic resource]'. - Mode of access: http://travel-world.pp.ua/ '] .isReference ():
print x.name + '(' + x.subjName, x.propName, x.objName + ') |'

The proposed method for constructing knowledge bases and expert systems in Python has advantages over existing ones. The main advantage is that the developer has access to all functions of the Python language. By expanding the functionality of the system, there is no need to invent another special language for describing the knowledge and query language for them. Thanks to the versatility of Python, the knowledge base is flexible for change; there is an easy way to improve classes, attributes, outbound rules and queries. These principles can be used to develop a full-fledged expert system in service delivery systems.

References

[1] Subbotin S.O. Presentation and processing of knowledge in systems of artificial intelligence and decision-making support: a manual CS. Zaporizhzhya: ZNTU, 2008. - 341 p. [in Ukrainian]

[2] Rybina G.V. Fundamentals of the construction of intelligent systems - Moscow: Finance and Statistics, INFRA-M, 2010. - 432 p. [in Russian]

[3] Beatles D. Python. Detailed directory - SPb .: Symbol-Plus, 2010. - 864 p. [in Russian]

[4] Lutz M. Programming on Python - SPb .: Symbol-Plus, 2002. – 1136. [in Russian]