Developing a Model for Optimizing Inventory of Repairable Items at Single Operating Base

Tin Le Vinh Trung

Bachelor’s thesis
May 2016
Technology, Communication and Transport
Degree Programme in Logistics Engineering

jamk.fi

Description

Author(s) Le, Vinh Trung Tin	Type of publication Bachelor's thesis	Date 30.05.2016
		Language of publication: English
	Number of pages 63	Permission for web publication: x
Title of publication **Developing a Model for Optimizing Inventory of Repairable Items at Single Operating Base**		
Degree programme Logistics Engineering		
Supervisor(s) Lehtola, Pasi		
Assigned by Lehtola, Pasi		

Abstract

The use of EOQ model in inventory management is popular. However, EOQ models has many disadvantages, especially, when the model is applied to manage repairable items. In order to deal with high-cost and repairable items, Craig C. Sherbrooke introduced a model in his book "Optimal Inventory Modeling of Systems: Multi-Echelon Techniques".

The research focus is to implement and develop a program to execute the single-site inventory model for repairable items. The model helps to significantly increase the availability of the system and decrease the inventory cost, in comparison to, the model for consumable items using EOQ (Economic Order Quantity) method.

The optimized stock level is calculated by using two different techniques, including: Marginal analysis technique in Excel and special algorithm in Python. After that, the result is compared to each other to validate the program.

Companies currently acquiring high tech or expensive systems can benefit from the research. The inventory model not only helps to reduce the inventory cost, but also increased the availability of the system. In addition, detailed guidelines for creating the model using Excel and Python is also a reference for companies who would like to apply this model to their operations.

.

Keywords/tags (subjects)
Inventory optimization, Poisson process, Python, marginal analysis, expected backorders, operational availability, fleet availability

Miscellaneous

CONTENTS

FIGURES

Tables

ABBREAVIATIONS

ALDT – Administrative and Logistics Delay Time

EBO – Expected Backorder

EOQ – Economic Order Quantity

MCMT – Mean Corrective Maintenance Time

MCT – Mean Corrective Time

MDT – Mean Down Time

MPMT – Mean Preventive Maintenance Time

MPT – Mean Preventive Time

MTBF – Mean Time between Failures

MTBPM – Mean Time between Preventive Maintenance

MTTR – Mean Time to Repair

NMCT – Non Mission Capable Time

1 INTRODUCTION

1.1 Background information

The introduction of Economic-Order-Quantity model in 1915 by Ford W. Harris has been greatly influencing the way to manage the inventory. Waters (2003, 259) describes EOQ model as flexible, easy-to-use and widely applied to inventory problems. In brief, EOQ model takes the information about the demand, unit cost, reorder cost and holding cost of one single item to compute the best order quantity, in order to avoid stock out. According to Sherbrooke (2004, 1-2), the EOQ model (also called the item approach) is simple because it only deal with one decision variable which is "when to order or the stock level". In addition, the stock level for an item is determined by a simple formula which balance the cost of holding inventory, ordering and stock out. Also, Sherbrooke (2004, 3) pointed out that the big disadvantage of EOQ model is that it determines the stock level of one item without considering other items in a system.

Without being satisfied with EOQ model when applying for repairable items, Craig C. Sherbrooke invented a new model to determine the stock level for those items which are expensive and repairable. This model is shown in the book "Optimal Inventory Modeling of Systems: Multi-Echelon Techniques (2nd edition)". Throughout the book, Craig C. Sherbrooke introduced the logic of his model and also the comparison to EOQ model in terms of the increase in availability and decrease in inventory holding cost.

In general, the book covers the development of the model in three steps. Firstly, a single-site model performs broken items is repaired at the base where the system is operated. Then, the model evolves to multi echelon, in which the broken items are not only repaired at site, but also at intermediaries or suppliers. Finally, the multi-indenture model divides items into class.

In this thesis, only the first stage of the model development at single site is taken into consideration. By imagination, a fleet of ten identical aircrafts is set as an input. Each of them has two critical items which occurred two times in one aircraft. The author would perform the model created by Craig C. Sherbrooke in Excel and constructed a program in Python to execute.

Beforehand, the author only intended to do the thesis in Excel, however, Excel showed some disadvantages which took more times for repetitive tasks and was not flexible when the input data changed. After that, the author has been studying a new programming language named Python for two and a half months, in order to construct the program. The reason for Python choice is that it is a high-level, powerful and easy-to-learn program, compared to C++ or Java. That is why it is very suitable for beginners who does not have any programming knowledge.

This thesis is theoretical and not assigned to any companies. However, companies currently acquiring and operating high tech/expensive systems could benefit from it. Also companies with the desire to build a program to manage their inventory can see the open source code in this thesis as a guideline.

1.2 Objectives

The first objective of the thesis is to successfully implement the model in Excel with assumed input data using the "Marginal analysis" technique introduced in the book. Then a separate program is constructed in Python using special algorithm (not the marginal analysis technique). Both ways of running Craig C. Sherbrooke single-site model have to give the same output with the same data input. Finally, the result from the program should show an increase in the availability and decrease in inventory cost, in comparison to EOQ model.

2 RESEARCH METHODS

2.1 Research methods in general

Kothari (2013, 1-2) states that the purpose of the research is to discovers answers to questions through the application of scientific procedures and it is the search of knowledge. The main aim or objective of a research is to find out the truth which is hidden or has not been discovered as yet (Kothari 2013, 2).

The basic types of research can be divided into five categories, including: Descriptive vs. analytical, applied vs. fundamental, quantitative vs. qualitative, conceptual vs. empirical and other types of research (Kothari 2013, 2-3). In the context of the thesis, it is worth to mention the applied vs. fundamental and quantitative vs. qualitative categories.

The applied research means a solution is found in order to solve practical problems arising in society, organizations, etc. On the contrary, fundamental research deals with the generalization and the formulation of a theory. Research questions from applied research could be "How to produce a product?" or "How to manage the inventory". According to fundamental research, example shall be "Analysis of factors affect each stage of product life cycle".

In general, there are two basic approaches for a research, namely qualitative and quantitative. The former approach is concerned with subjective assessment of attitudes, opinions and behaviors (Kothari 2013, 4). Thus qualitative approach is used to understand the underlying reasons or motivations of a phenomena. Regularly, qualitative research gives more insight into the problems. With reference to quantitative approach, it is based on the quantitative measurements of some characteristics (Kothari 2013, 3). To be more specific, quantitative approach deals with numerical data by quantifying problem, then the data will be treated using statistical procedure to give informative conclusion.

2.2 The scope of the thesis

This thesis is an applied research because a solution to manage the inventory for repairable items is found. Here it is a program in Python which can be applied to companies holding expensive systems to achieve higher availability and decrease inventory cost.

The approach of this thesis is both qualitative and quantitative. The reason is that the thesis is explanatory and exploratory. The explanatory characteristic is illustrated through the review process of the Craig C. Sherbrooke model development. Specifically, it gives an insight understanding about the way to perform the model in real life. For exploratory attribute, the constructed program helps to explore a new way to build Craig C. Sherbrooke model in practice.

The data collection method in the thesis is not as usual for qualitative and quantitative approach. In fact, there is not any interviews, questionnaires, telephone calls or historical data. The reason is that the thesis is theoretical and acted as a general development project of a product. In this case, the product is the software code. The data here is not collected from the real life, but, generated on the purpose of users and the authors. For example, users can change the data input in the program if they would like the fleet has 5, 20 and so on aircrafts. Also the number of critical items and their occurrences can be adjusted.

3 THEORETICAL BASIS

3.1 Inventory

The term inventory has several definitions. Waters (2003, 252) defines inventory as a list of things hold in stock. Arnold (2012, 268) goes a little deeper into the definition by state that Inventories are materials and supplies that a business or institution carries either for sale or to provide inputs or supplies to the production process.

According to Arnold (2012, 269-270), inventory is classified in to five categories: raw materials, work-in-process, finished goods, distribution inventories and MROs (Maintenance, repair and operational supplies). In the thesis context, the type of inventory is spare part and it is also a critical item in a system.

3.2 Consumable vs. repairable items

Consumables are goods that are capable of being consumed, that maybe destroyed, dissipated, wasted or spent (Wikipedia 2016). Whereas, repairable items are those could be repaired when malfunction exists. In fact, repairable items tend to be expensive and the demand for any particular item tends to be low (Sherbrooke 2004, 6).

3.3 EOQ Model

Economic Order Quantity is the simplest way to determine the size of an order. It is the found at the balance between the cost of placing an order and the cost of holding it. EOQ makes assumptions that there will be no stock-outs, zero lead times and that we can 'safely' order when at zero stocks (Emmett 2005, 59.)

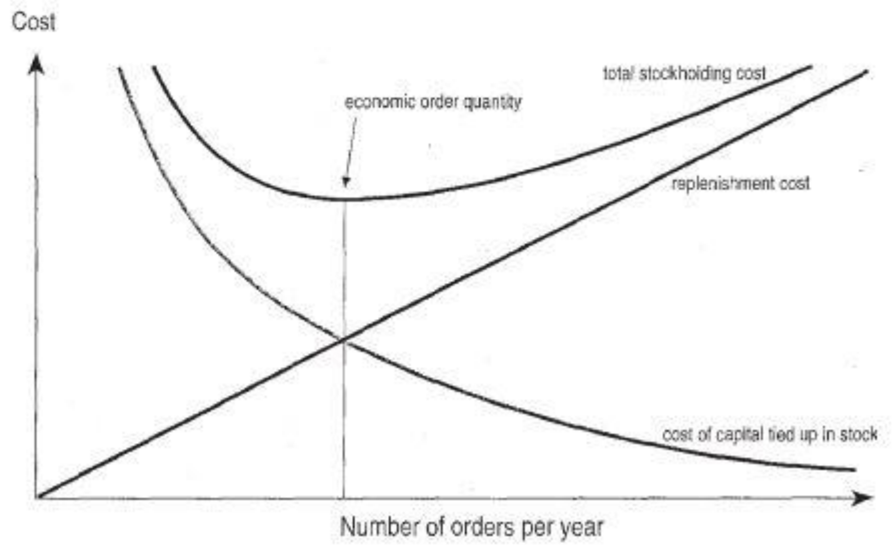


Figure 1. Economic Order Quantity (Adapted from Emmett 2005, 60)

Economic Order Quantity formula is performed as:

$$Q = \sqrt{\frac{2 * \Omega * m}{i * c}}$$

Where Q is the economic quantity, m is the mean annual demand, Ω is the cost to place an order, i is the annual holding cost rate and c is the unit cost of the item.

3.4 Poisson distribution

Considering X be the discrete random variable representing the number of random events in a specified time. The Poisson probability mass function is in the form:

$$p(x) = (m * T)^x * \frac{e^{-m*T}}{x!} \; with \; x = 0, 1, 2, \ldots$$

Where m is the average annual demand and T is the average time. The Poisson probability mass function gives the probability that x failures occur during a certain time.

3.5 Poisson process

If failures have Poisson distribution with λ equal to m*T. The time between those failures is exponentially distributed with parameter λ. Let Y_k be a random variable representing the time of the kth failure with $Y_k = \sum_{i=1}^{k} T_i$ where T_i is the time between failure i-1 and failure i. Then the sum of k independent exponential random variables has a gamma distribution with parameters k and λ (Ebeling 1996, 52). The cumulative distribution function for Y_k can be written as:

$$\Pr\{Y_k \leq t\} = 1 - e^{-\lambda t} \sum_{i=0}^{k-1} \frac{(\lambda t)^i}{i!}$$

The Poisson process is often applied in inventory analysis to determine the number of spare components when the time between failures is exponential.

$$R_s(t) = \sum_{n=0}^{S} p_n(t)$$

The above formula is the cumulative probability of S spares or fewer failures occurring during time t. It represents the probability of satisfying all demands for spare components during time t (Ebeling 1996, 53.)

A fundamental result can be concluded:

$$P(Y_k \leq t) = P(Number\ of\ failures \geq s)$$

3.6 Palm's theorem

According to the Palm's Theorem, the steady state probability distribution for the number of units in repair has a Poisson distribution with mean **m*T** if the demand for and item is a Poisson process with annual mean **m** and the repair time for each failed unit is independently and identically distributed according to any distribution with mean **T** years (Sherbrooke 2004, 22.)

Reflecting to the thesis, Palm's theorem helps calculate the number of items in repair by simple formula without need for collecting the data to find the shape of repair distribution.

3.7 Stock level

In this scope of the thesis, the stock level means the number of spare items purchased beforehand. In the single site model, each failed unit is repaired immediately with mean time **T**. The assumption is that all of critical items can be fixed without being thrown away.

Based on the assumption, there are some incidents where the spare units are on the shelf, or they will be in repaired process. There also cases when there is no spare available on the shelf and this causes backorders when demand happens. Finally, a stock balance equation has been created for the analysis.

Stock balance equation is in this form (Sherbrooke 2004, 24):

$$s = OH + DI - DO$$

According to the above formula, s is the stock level, OH is the number of spare units on the shelf, DI is the number of spare units undergoing repair and BO is number of backorders. Since the stock level is constant, the number of spare units in repair increase by one when one demand occur. When the repair is finished, the due in decrease by one and the number of spare items on shelf increase by one. In case all of spare units are in repair and demand still occurs, the number of backorders will increase.

3.8 Expected backorders

Expected backorders is an item performance measure which indicate the number of unfulfilled demand at a point in time. To be more specific, a backorder happens whenever a demand cannot be filled. The formula for calculating is as follow:

Expected backorder is calculated based on this formula (Sherbrooke 2004, 26):

$$EBO(s) = \Pr\{DI = s + 1\} + 2\Pr\{DI = s + 2\} + 3\Pr\{DI = s + 3\} + \cdots = \sum_{x=s+1}^{\infty} (x - s) * \Pr\{DI = x\}$$

Where $\Pr\{DI = s + 1\}$ is the probability that the “s+1” number of items are in repaired and x is the stock level “s” plus number of backorders. In general, the formula calculate the expected value of backorders by summing all of the product of number of backorders occur and the probability for that amount of backorder happens. However, this formula makes the computation in Excel and Python program difficult because there is no endpoint for the number of backorder would occur. That is why a transformation of this formula is needed.

General expression deduction of EBO with Poisson distribution is performed as follows (JFukuda 2007):

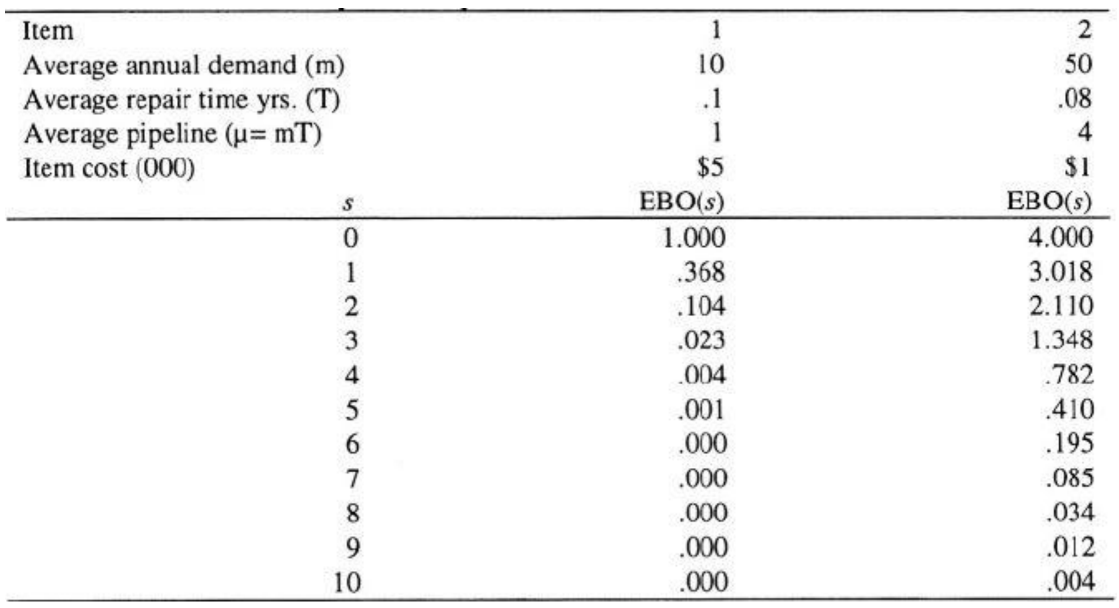
$$EBO(s) = PL * Poisson(s, PL) + (PL - s) * [1 - PoissonAc(s, PL)]$$

Where PL is the average pipeline (Average pipeline = **m*T**), Poisson(s,PL) is the probability of exactly s spares to occur with specified average pipeline and PoissonAc(s,PL) is the cumulative of Poisson distribution from 0 to s spare unit. As can be seen from the formula, the calculation for Poisson(s,PL) and PoissonAc(s,PL) can now be computed easily in Excel and Python.

3.9 Marginal analysis

Business Dictionary (2016) defines marginal analysis as a process of identifying the benefits and costs of different alternatives by examining the incremental effect on total revenue and total cost caused by a very small change in the output or input of each alternative. This explanation may come from microeconomics or accounting point of view, but the general idea is the same.

Table 1. Numerical example for single-site model (Adapted from Sherbrooke 2004, 30)



Item	1	2
Average annual demand (m)	10	50
Average repair time yrs. (T)	.1	.08
Average pipeline (μ= mT)	1	4
Item cost (000)	\$5	\$1
s	EBO(*s*)	EBO(*s*)
0	1.000	4.000
1	.368	3.018
2	.104	2.110
3	.023	1.348
4	.004	.782
5	.001	.410
6	.000	.195
7	.000	.085
8	.000	.034
9	.000	.012
10	.000	.004

An example taken from Craig C. Sherbrooke book illustrates a unit (an aircraft) comprises of two critical items with given input. In order to determine how many stock should be procured for each items, a decision of choosing either critical item 1 or 2 for each stock level is made by using the below formula (Sherbrooke 2004, 31).

$$\frac{[EBO(s-1) - EBO(s)]}{c}$$

Where EBO (s-1) is the expected backorder at “s-1” stock level, EBO (s) is the expected backorder at “s” stock level and c is the unit cost of a specified critical item. This formula calculates the marginal decrease in expected backorders divided by the item cost (Sherbrooke 2004, 31). If the result from the formula of item 2 is greater than item 1 at the same stock level, then item 2 will be chosen. The process will continue until the total amount of money invested in spare items approach the limit of maximum available inventory cost.

3.10 Availability

The calculation of availability has three different versions, namely inherent availability, achieved availability and operational availability. The inherent availability is the simplest out of three calculations. It determines the percentage of time that a system will theoretically be available or work properly. Below is the formula for inherent availability (Jones 2006, 10.2).

$$A_i = \frac{MTBF}{MTBF + MTTR}$$

The achieved availability is an improvement from the inherent availability because it takes into consideration the MTBM rather than MTBF. The MTBM may be smaller than MTBF, because it makes allowance for periods when the system will not be available due to preventive maintenance activities or when maintenance will not be performed but a failure may have occurred . Below is the formula for achieved availability (Jones 2006, 10.4.)

$$A_a = \frac{MTBM}{MTBM + MCMT + MPMT}$$

The last method of calculating availability is called operational availability. It is the percentage of time when under actual operating conditions the system can perform its mission (Jones 2006, 10.5). The formula for operational availability has slightly difference from two books.

$$A_o = \frac{MCT}{MCT + NMCT}$$

Where NMCT = MCT + MPT + ALDT (Jones 2006, 10.6)

$Operational\ Availability = \frac{100 \times MTBM}{MTBM + MDT}$ (Sherbrooke 2004, 38)

In the scope of this thesis, the availability means the fleet availability which is the expected percent of the aircraft fleet that is not down for any spare and is given by this formula (Sherbrooke 2004, 39):

$$A = 100 \times \prod_{i=1}^{I} \{1 - \frac{EBO_i(s_i)}{NZ_i}\}^{Z_i}$$

Where Z_i is the number of occurrences of ith critical item, N is the fleet size (Number of aircrafts in a fleet) and $EBO_i(s_i)$ is the expected backorder of ith critical item at ith stock level.

3.11 Python

3.11.1 General Information

In brief, Python is a programming language like JavaScript, C++, etc. It was invented by Guido van Rossum who is a programmer from the Netherlands and implemented in 1989 (Wikipedia 2016).

Python is known as a very powerful and language due to these reasons (Python Wiki 2014):

- Equipped with elegant syntax, so it is easy to read and use. Consequently, it is suitable for beginners who do not have background in programming.
- Supported by considerable libraries for programming tasks. In addition, developers around the world have created prepared modules (data analysis, statistics, simulation, etc.) and users just need to download and embed it to Python without constructing the module again.
- Python can be executed on my operating systems: Windows, MacOS, Unix, etc.
- Finally, Python has the open source license which means it is totally free of charge and users can freely edit and redistribute.
- In comparison to Java, Python is slower to run, however, it requires less time to develop, approximately 3-5 times. In addition, Python can also act as a glue language which means a program can be constructed in Java and then combined to Python (Python Software Foundation 2016).

3.11.2 Variables

Severance (2013, 20) stated that the most powerful features of programming language is the ability of manipulating variables. To be more specific, a variable is a name which contains value. Scientifically, memory space is reserved when variables are created (Tutorials Point 2016).

In Python, users do not need to declare variables when assigning a value to it, whereas Java require to do so. A value is assigned to a variable name by using "=" operator. The variable name is located on the left and its value is placed on the right (Tutorials Point 2016).

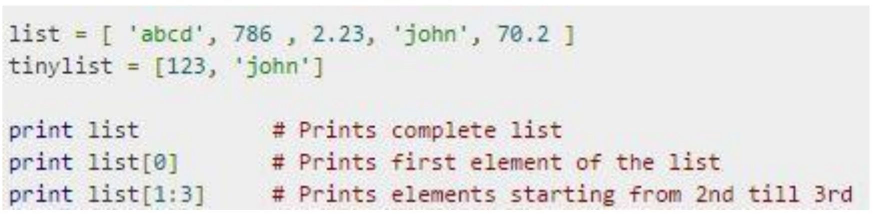
3.11.3 Variable types

The type of the value assigned to the variable varies. Basically, there are five standard types, including: number, string, list, tuple and dictionary. In the program of this thesis, only tuple type was not used.

The number data type is numeric values which comprises 4 categories in Python, including: integers (-1, 2, 10, etc.), long integers (integers performed in octal and hexadecimal), floating numbers or number with decimals (3.56, 5.7, etc.) and complex numbers (6+8i where I is the imaginary unit) (Tutorials Point 2016.)

The string type is defined as contiguous set of characters represented in quotation marks (Tutorials Point 2016). In practice, a string can contain both words and values but it has to be put inside single or double quotation marks in Python. For example, a number placed inside quotation marks which is assigned to a variable will be defined as string type (not number type) in Python.

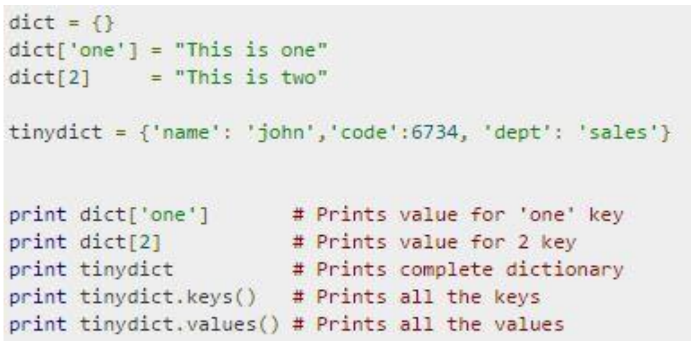
A list is a sequence containing elements. Values or elements in a list can be any type (integers, floating integers, words, etc.) and enclosed in square brackets. One important characteristic of a list is that it is mutable, one can add, delete, slice or even combine two lists into one. Below is an example of retrieving elements in a defined list by using slice operators ([]) in Python.



```
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tinylist = [123, 'john']

print list          # Prints complete list
print list[0]       # Prints first element of the list
print list[1:3]     # Prints elements starting from 2nd till 3rd
```

Figure 2. Example of list in Python (Adapted from Tutorials Point 2016)

A dictionary is pretty similar to a list which contains a set of elements. However, it is a combination of a so-called key-value pair. Like in a real dictionary, there are words in one language and a simultaneous explanation in another one. Another difference between a list and a dictionary is that elements inside a list is in ordered, whereas an element in a dictionary is navigated by the key name. Here is an example

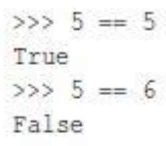


```
dict = {}
dict['one'] = "This is one"
dict[2]     = "This is two"

tinydict = {'name': 'john','code':6734, 'dept': 'sales'}


print dict['one']       # Prints value for 'one' key
print dict[2]           # Prints value for 2 key
print tinydict          # Prints complete dictionary
print tinydict.keys()   # Prints all the keys
print tinydict.values() # Prints all the values
```

Figure 3. Example of dictionary in Python (Adapted from Tutorials Point 2016)

3.11.4 Conditional execution

Conditional execution means outputs will be printed in Python depending on the in-puts and conditional expressions. The first type of conditional expression is called Boolean. A Boolean expression is an expression which only gives two results (either True or False). The Boolean expression comprises one operator and two operands. The operator have 7 types, including: != (not equal), > (greater than), >= (greater than or equal to), < (less than), <= (less than or equal to), is (the same as) and is not (not the same as).



```
>>> 5 == 5
True
>>> 5 == 6
False
```

Figure 4. Boolean expression in Python (Adapted from Severance 2013, 31)

When there are more conditions (Two or more Boolean expressions), a logical operator needs to be used. Three typical types of logical operators are and, or and not. For example, an input x has the condition (x > 5 and x <10) and Python will print the result True if any value of x is within that range, otherwise the result is False.

In order to make the program useful, users need the ability to check conditions and change the behavior of the program accordingly. Severance (2013, 32). In Python, “if” statement accompanying with colon represents for conditional statements.

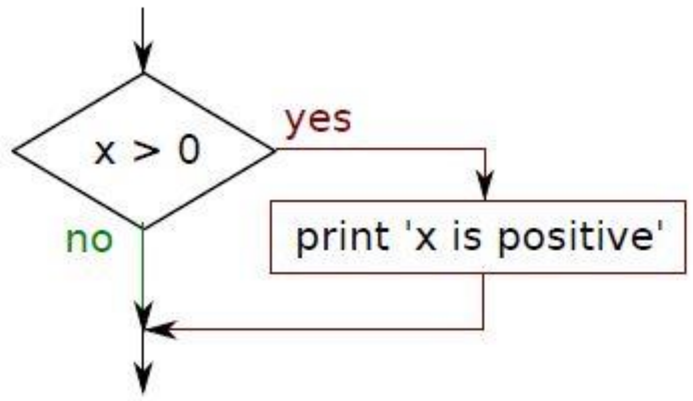


Figure 5. Conditional Execution (Adapted from Severance 2013, 32)

The second type of if statement is called alternative execution. Alternative execution has two possibilities and the condition will determine which one gets executed (Severance 2013, 32).

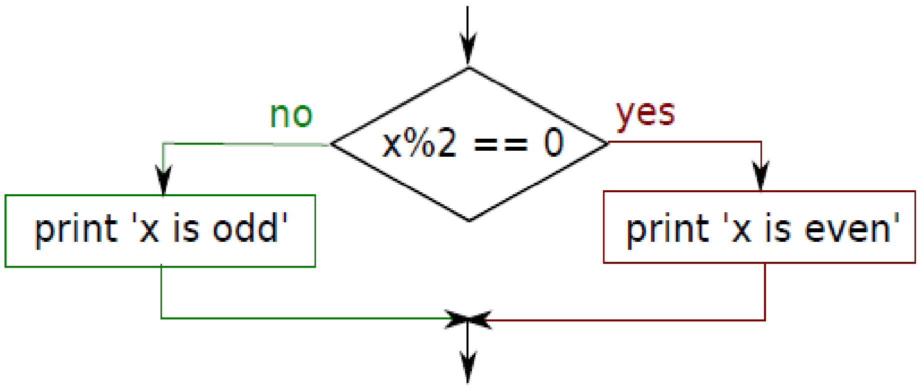


Figure 6. Alternative Execution (Adapted from Severance 2013, 33)

With more than two possibilities, a chained conditional need to be applied. In Python, “elif” statement will add more conditions and there is no limit of elif statement. The condition chain will be closed by the “else” statement.

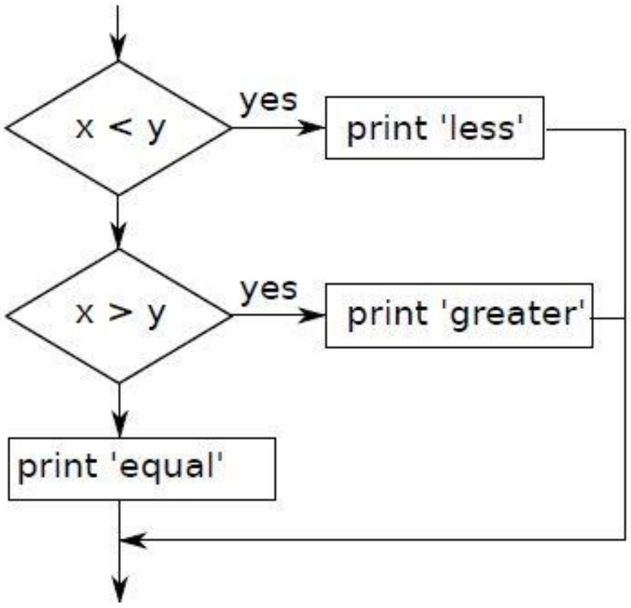


Figure 7. Chain conditional (Adapted from Severance 2013, 34)

The final type of conditional execution is nested conditionals which nest one condition to the other. Below is an example of the syntax and the structure of nested conditional.

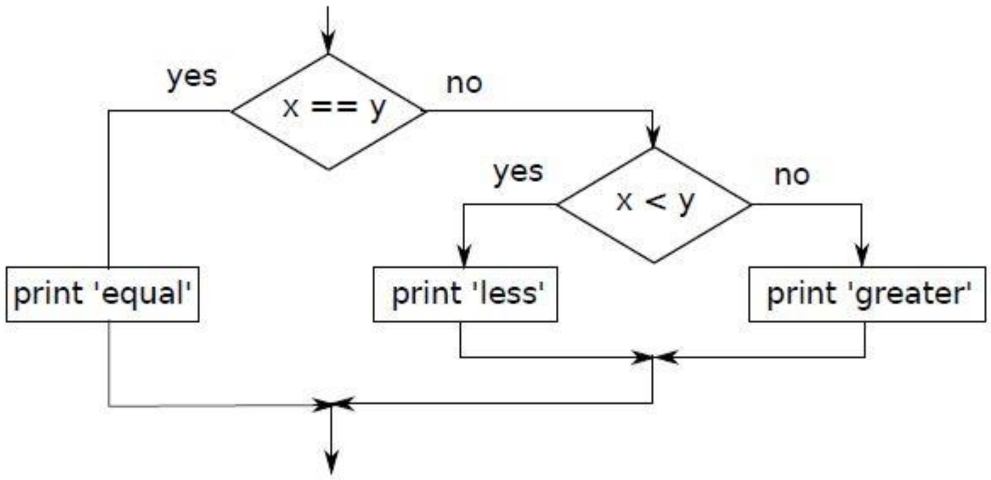
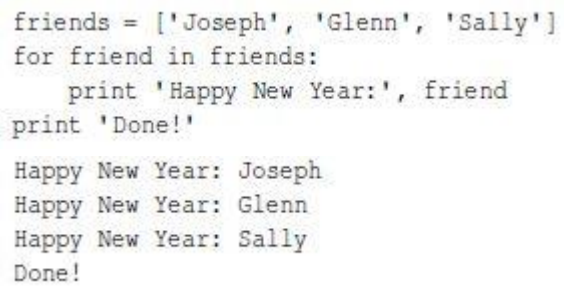


Figure 8. Nested conditional (Adapted from Severance 2013, 35)

3.11.5 Repetition

When users would like to repeat some statements for fixed number of time or until there is a signal to make the repetition to stop. In Python, the repetition is performed through loop. There are two types of loop which are definite an infinite loops.

A definite loop is used to iterate values in a sequence or a range. A definite loop in Python has the general form “for <var> in <sequence>”. The name <var> is a loop-counter or loop-index variable. It will be assigned a value in the sequence and some actions will be performed for every iteration.

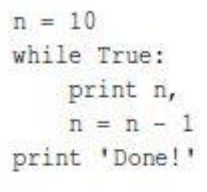


```
friends = ['Joseph', 'Glenn', 'Sally']
for friend in friends:
    print 'Happy New Year:', friend
print 'Done!'
```

```
Happy New Year: Joseph
Happy New Year: Glenn
Happy New Year: Sally
Done!
```

Figure 9. Definite loop example in Python (Adapted from Severance 2013, 60-61)

On the contrary to definite loop, infinite loop does not have any iteration variable which gives signal to the loop when to stop. The infinite loop is controlled by ”while” statement in Python.



```
n = 10
while True:
    print n,
    n = n - 1
print 'Done!'
```

Figure 10. Infinite loop syntax in Python (Adapted from Severance 2013, 59)

As can be seen from the above example, there is no signal for the program to stop and Python will run this program forever till minus infinity.

4 BUILDING THE MODEL BY USING EXCEL

In this chapter, all needed steps for building the model to find optimal spares in Excel are illustrated. Also applied formulas and Excel functions are mentioned. For simplicity, each unit comprises only two critical items and each item occur two times. Finally, there are 10 units in the fleet (Fleet size is equal to 10).

4.1 Taking inputs and calculating the average pipeline

The inputs needed for the model has to be filled for every items in one unit, except the fleet size. They include:

- Average annual demand (m): the term demand here is similar to failures. When a failure occurs, a demand for a spare is needed.
- Average repair time in years (T): the mean time to fix one item. In the model, this time should be converted to year unit because the average annual demand was also formatted in years.
- Item Cost (000 $): The price of one item. The unit is thousand dollars.
- Occurrences: The number of locations for one type of item on a unit.
- Fleet size (N): Number of units in a fleet.

After collecting needed inputs, the average pipeline μ was calculated by applying Palm's theorem. The formula is $\boldsymbol{\mu = m * T}$.

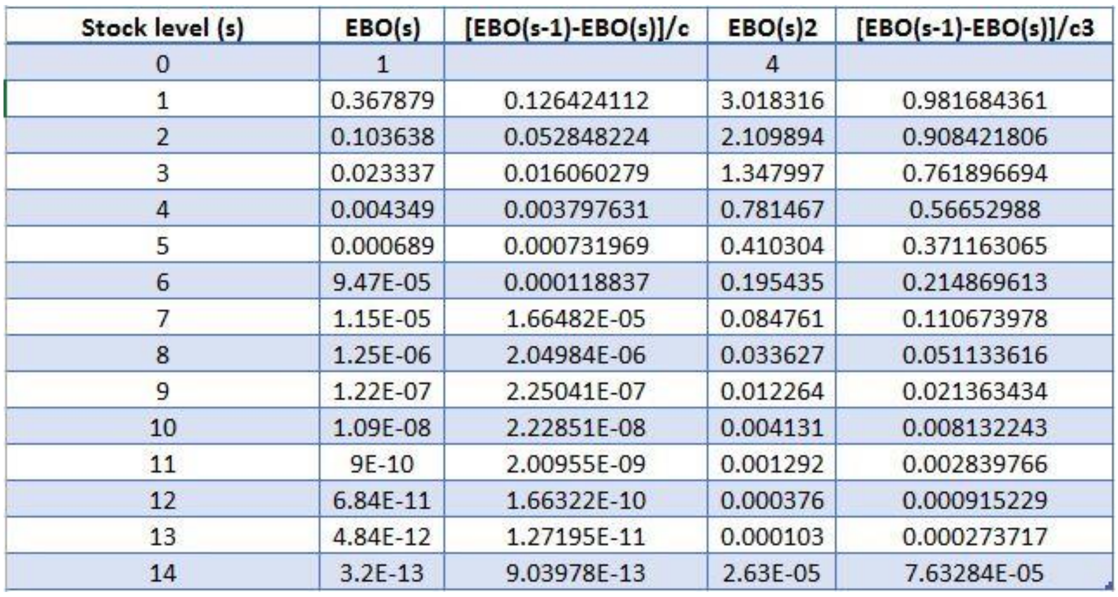
4.2 Computing EBOs and marginal values of EBOs for each stock level

By applying the deduction formula for EBOs calculation, EBOs' values were generated corresponding to stock levels. The formula was shown in section 3.8 (JFukuda 2007).

In Excel, Poisson(s, PL) and PoissonAc(s, PL) was computed by using Poisson.dist fuction. The difference between those is at the third argument. The input of 0 (probability mass function) was applied for the former, whereas, the latter had the input of 1.

After finishing the EBOs calculation, the next step is to perform the marginal value when increasing the stock level by one unit. Basically, it is equal to the difference between the previous and the current EBOs divided by the item cost.

Table 2. Calculation of marginal value in Excel



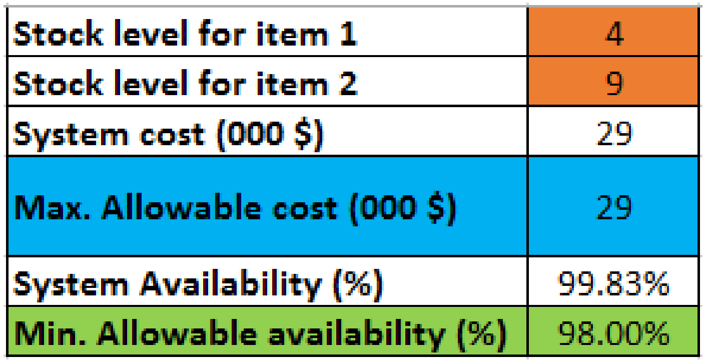
Stock level (s)	EBO(s)	[EBO(s-1)-EBO(s)]/c	EBO(s)2	[EBO(s-1)-EBO(s)]/c3
0	1		4	
1	0.367879	0.126424112	3.018316	0.981684361
2	0.103638	0.052848224	2.109894	0.908421806
3	0.023337	0.016060279	1.347997	0.761896694
4	0.004349	0.003797631	0.781467	0.56652988
5	0.000689	0.000731969	0.410304	0.371163065
6	9.47E-05	0.000118837	0.195435	0.214869613
7	1.15E-05	1.66482E-05	0.084761	0.110673978
8	1.25E-06	2.04984E-06	0.033627	0.051133616
9	1.22E-07	2.25041E-07	0.012264	0.021363434
10	1.09E-08	2.22851E-08	0.004131	0.008132243
11	9E-10	2.00955E-09	0.001292	0.002839766
12	6.84E-11	1.66322E-10	0.000376	0.000915229
13	4.84E-12	1.27195E-11	0.000103	0.000273717
14	3.2E-13	9.03978E-13	2.63E-05	7.63284E-05

4.3 Setting constraints and finding optimal policy

For the model, there are two types of constraints, including: Maximum allowable cost for spares and minimum allowable availability. User can choose one of those constraints and find the optimal spares for each item.

In the example shown below, the cost constraint was set for 29,000 dollars. By applying the marginal analysis, we first chose 6 spares for item 2. At this point, the cost was just 6,000 dollars, so the stock level had to be continued to increase. However, the next choice was for item 1 with higher marginal value (0.126) compared to (0.111) of item 2 (s = 7). With the same logic, the optimal spares were 4 and 9 for item 1 and 2, respectively. According to these option, the system cost was 29,000 $ which equaled to the maximum allowable cost for spares. Finally, the system availability is 99.83 %.

Table 3. Optimal stock level result in Excel



Stock level for item 1	4
Stock level for item 2	9
System cost (000 $)	29
Max. Allowable cost (000 $)	29
System Availability (%)	99.83%
Min. Allowable availability (%)	98.00%

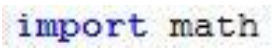
In order to pick the value of EBOs based on the input of stock level for item, a combination of MATCH and INDEX function are used. MATCH function helps to locate the row of targeted stock level, then INDEX function gives the value in EBOs column corresponding to the certain stock level.

5 CONSTRUCTING A PROGRAM TO RUN THE MODEL BY USING PYTHON

5.1 Step 1 – Importing libraries and creating base variables

5.1.1 Importing libraries

The use of factorial and exponential function for calculating EBOS in Python require the import of math library. This library helps the program to calculate the value of Poisson distribution.

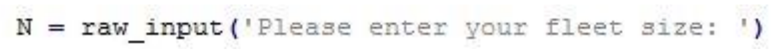


```
import math
```

Figure 11. Importing library in Python

5.1.2 Creating base variables

Three base variables were created, including: N (Fleet size), numitems (Number of critical items in one unit) and length (Number of stock levels taken into consideration). Those variables are created by using “raw_input()” function to take the data from users and the format is always a string. In order to use the user’s input data as an integer for calculation, it has to be converted to integer or real numbers by using “int()” or “float()” function.

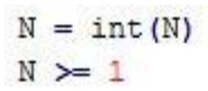


```
N = raw_input('Please enter your fleet size: ')
```

Figure 12. Asking the user for input syntax

5.1.3 Conditions for base variables

For base variables, it is really important to get the right format of input from users. To be more specific, the fleet size (The number of aircrafts or units) have to be positive integer. The same rule applies to the number of critical items that a unit comprises of and the number of stock levels.



```
N = int(N)
N >= 1
```

Figure 13. Conditions for base variables syntax

5.1.4 Prevention for errors

In order to prevent users from accidentally or intentionally typing wrong required inputs (1 instead of “one”), a combination of infinite loop and try/except function was created. The idea of try/except function is to return a notice line to warn user, in case they type a wrong input. With reference to infinite loop, It helps user to retype the data without re-running the program again.

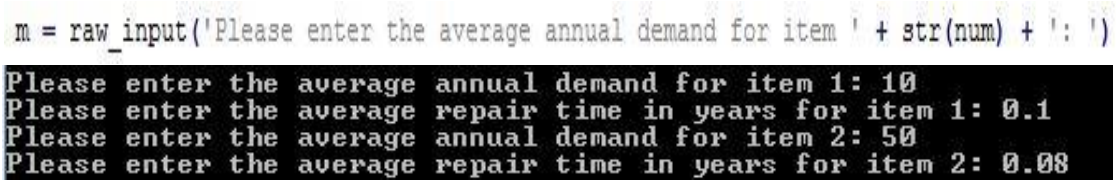
5.2 Step 2 - Creating a list of EBOs

5.2.1 Adding additional inputs

EBO is calculated based on the general expression deduction of EBO formula (Section 3.8). Firstly, we need to define the average pipeline (apl) variable by taking the multiplication of the average annual demand (m) and repair time (T). For those two components of the average pipeline, the code was made in a way so that the user can input the data for each critical items step by step. Actually, it is made by applying finite loop and range() function. The range function helps to create a list of item numbers which is equal to the number of critical items in a unit. Then a temporary variable can loop inside the list.

5.2.2 Calculating EBOs

According to EBO formula shown above, the average pipeline variable for each item is already calculated and defined. However, Python does not know how to compute the Poisson pdf and cdf, except importing another library. Fortunately, the Poisson pdf can be calculated by using the formula in section 3.8 (JFukuda 2007).

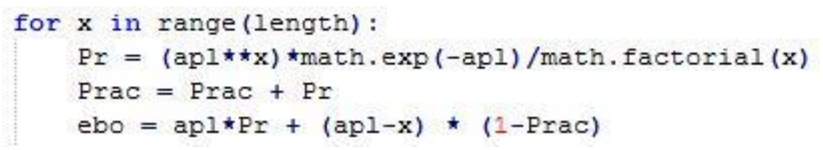


```
 m = raw_input('Please enter the average annual demand for item ' + str(num) + ': ')
```

```
Please enter the average annual demand for item 1: 10
Please enter the average repair time in years for item 1: 0.1
Please enter the average annual demand for item 2: 50
Please enter the average repair time in years for item 2: 0.08
```

Figure 14. Asking for inputs for EBO calculation in Python

For calculating the cumulative Poisson distribution, a temporary variable is used to memorize the value of the Poisson pdf at stock level s = 0. Then, its value will be added by the Poisson probability at s = 1 after the second loop. This finite loop will end until it reaches the length of the desired number of stock levels.

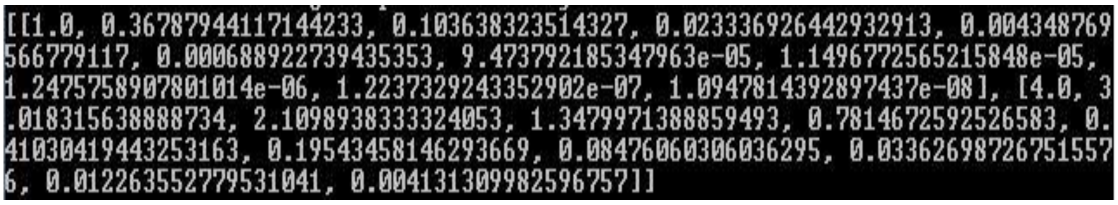


```
for x in range(length):
    Pr = (apl**x)*math.exp(-apl)/math.factorial(x)
    Prac = Prac + Pr
    ebo = apl*Pr + (apl-x) * (1-Prac)
```

Figure 15. Definite loop syntax for EBO calculation in Python

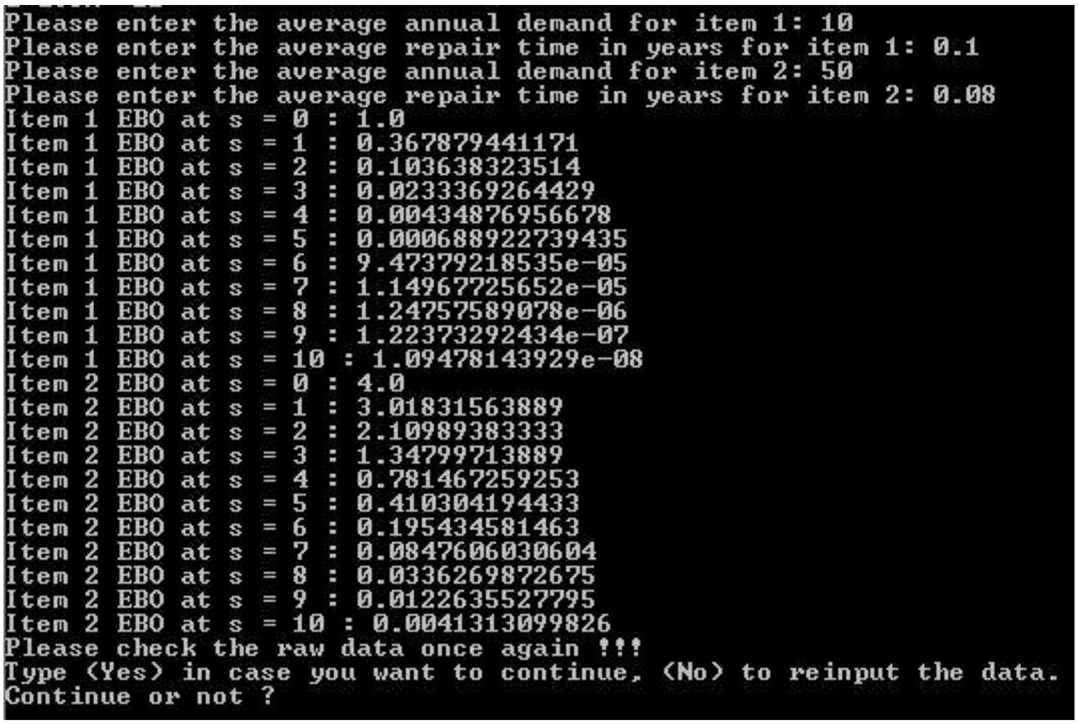
5.2.3 Adding EBOs to a list

At first hand, an empty list was created and named ”t” in Python. After each loop, the ebo variables will be added to the list in order by using ”t.append” function.



```
[[1.0, 0.36787944117144233, 0.103638323514327, 0.023336926442932913, 0.004348769
566779117, 0.000688922739435353, 9.473792185347963e-05, 1.1496772565215848e-05,
1.2475758907801014e-06, 1.2237329243352902e-07, 1.0947814392897437e-08], [4.0, 3
.018315638888734, 2.1098938333324053, 1.3479971388859493, 0.7814672592526583, 0.
41030419443253163, 0.19543458146293669, 0.08476060306036295, 0.03362698726751557
6, 0.012263552779531041, 0.004131309982596757]]
```

Figure 16. A list of EBOs in Python



```
Please enter the average annual demand for item 1: 10
Please enter the average repair time in years for item 1: 0.1
Please enter the average annual demand for item 2: 50
Please enter the average repair time in years for item 2: 0.08
Item 1 EBO at s = 0 : 1.0
Item 1 EBO at s = 1 : 0.367879441171
Item 1 EBO at s = 2 : 0.103638323514
Item 1 EBO at s = 3 : 0.0233369264429
Item 1 EBO at s = 4 : 0.00434876956678
Item 1 EBO at s = 5 : 0.000688922739435
Item 1 EBO at s = 6 : 9.47379218535e-05
Item 1 EBO at s = 7 : 1.14967725652e-05
Item 1 EBO at s = 8 : 1.24757589078e-06
Item 1 EBO at s = 9 : 1.22373292434e-07
Item 1 EBO at s = 10 : 1.09478143929e-08
Item 2 EBO at s = 0 : 4.0
Item 2 EBO at s = 1 : 3.01831563889
Item 2 EBO at s = 2 : 2.10989383333
Item 2 EBO at s = 3 : 1.34799713889
Item 2 EBO at s = 4 : 0.781467259253
Item 2 EBO at s = 5 : 0.410304194433
Item 2 EBO at s = 6 : 0.195434581463
Item 2 EBO at s = 7 : 0.0847606030604
Item 2 EBO at s = 8 : 0.0336269872675
Item 2 EBO at s = 9 : 0.0122635527795
Item 2 EBO at s = 10 : 0.0041313099826
Please check the raw data once again !!!
Type (Yes) in case you want to continue, (No) to reinput the data.
Continue or not ?
```

Figure 17. EBO for each stock level and item result

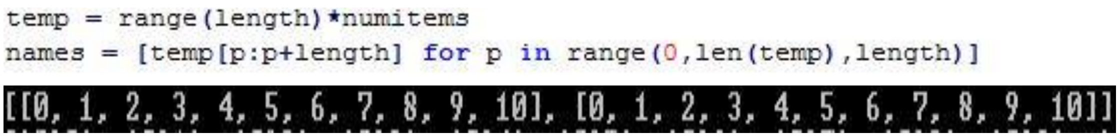
5.2.4 Prevention of errors

At this point, the program had run quite a lot. So after the input and calculations were displayed. A notice line will be shown and user has a chance to check the input for the EBOs calculation again before moving to the next step. In case of errors, user just needs to type ”No” and re-input the data again (for only this step).

5.3 Step 3 – Creating a list of all combined stock levels

5.3.1 Making a list with ordered numbers for each item

This step is like a buffer to help construct a list for combination. The below two-code lines help to construct a list with sublists inside for every item. Within one sublist, there are numbers start from 0 to the maximum number of stock levels (numitems variable).



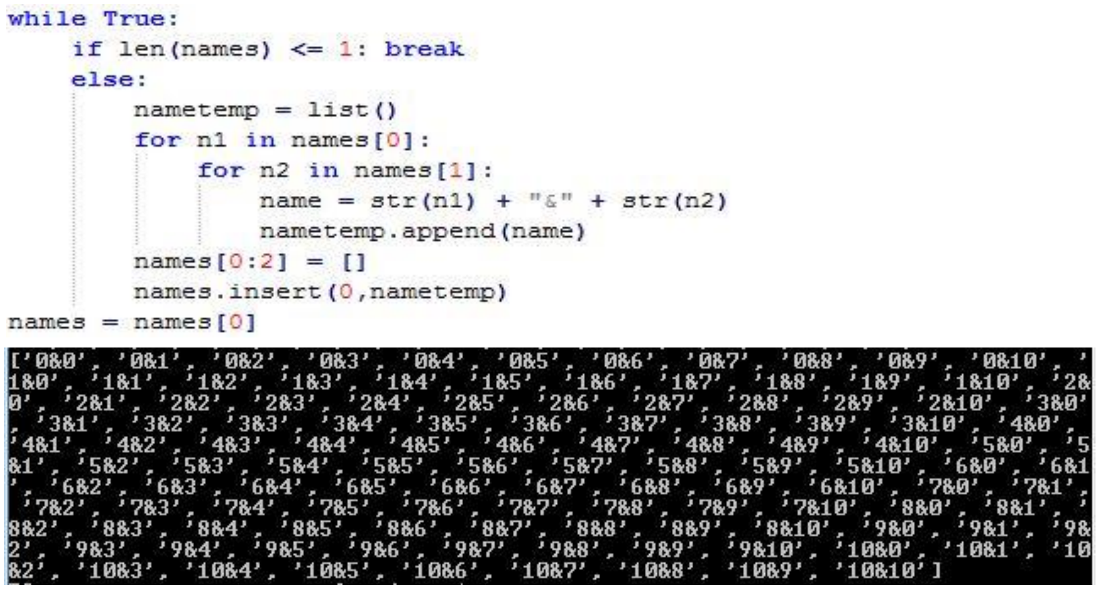
```
temp = range(length)*numitems
names = [temp[p:p+length] for p in range(0,len(temp),length)]
```

```
[[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10], [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]]
```

Figure 18. Making a list with ordered numbers for each item

5.3.2 Making a list for combination of stock level

After making a list of ordered number of stock levels, a two definite loop in the same list above was used. The result will be a general list for the names of each combination.



```
while True:
    if len(names) <= 1: break
    else:
        nametemp = list()
        for n1 in names[0]:
            for n2 in names[1]:
                name = str(n1) + "&" + str(n2)
                nametemp.append(name)
        names[0:2] = []
        names.insert(0,nametemp)
names = names[0]
```

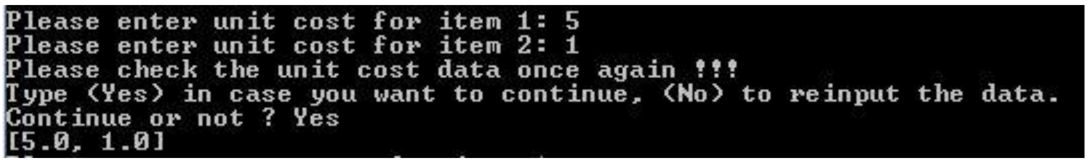
```
['0&0', '0&1', '0&2', '0&3', '0&4', '0&5', '0&6', '0&7', '0&8', '0&9', '0&10', '
1&0', '1&1', '1&2', '1&3', '1&4', '1&5', '1&6', '1&7', '1&8', '1&9', '1&10', '2&
0', '2&1', '2&2', '2&3', '2&4', '2&5', '2&6', '2&7', '2&8', '2&9', '2&10', '3&0'
, '3&1', '3&2', '3&3', '3&4', '3&5', '3&6', '3&7', '3&8', '3&9', '3&10', '4&0',
'4&1', '4&2', '4&3', '4&4', '4&5', '4&6', '4&7', '4&8', '4&9', '4&10', '5&0', '5
&1', '5&2', '5&3', '5&4', '5&5', '5&6', '5&7', '5&8', '5&9', '5&10', '6&0', '6&1
', '6&2', '6&3', '6&4', '6&5', '6&6', '6&7', '6&8', '6&9', '6&10', '7&0', '7&1',
 '7&2', '7&3', '7&4', '7&5', '7&6', '7&7', '7&8', '7&9', '7&10', '8&0', '8&1', '
8&2', '8&3', '8&4', '8&5', '8&6', '8&7', '8&8', '8&9', '8&10', '9&0', '9&1', '9&
2', '9&3', '9&4', '9&5', '9&6', '9&7', '9&8', '9&9', '9&10', '10&0', '10&1', '10
&2', '10&3', '10&4', '10&5', '10&6', '10&7', '10&8', '10&9', '10&10']
```

Figure 19. Making a list for combination of stock level

5.4 Step 4 - Creating cost list

5.4.1 Making a list for unit cost of every critical item

Firstly, an empty list called ”costdata” was builded up. Then this list will be added by the unit cost input from the user. From below extracted program display, the unit cost for item 1 and 2 is 5, 1 respectivley.

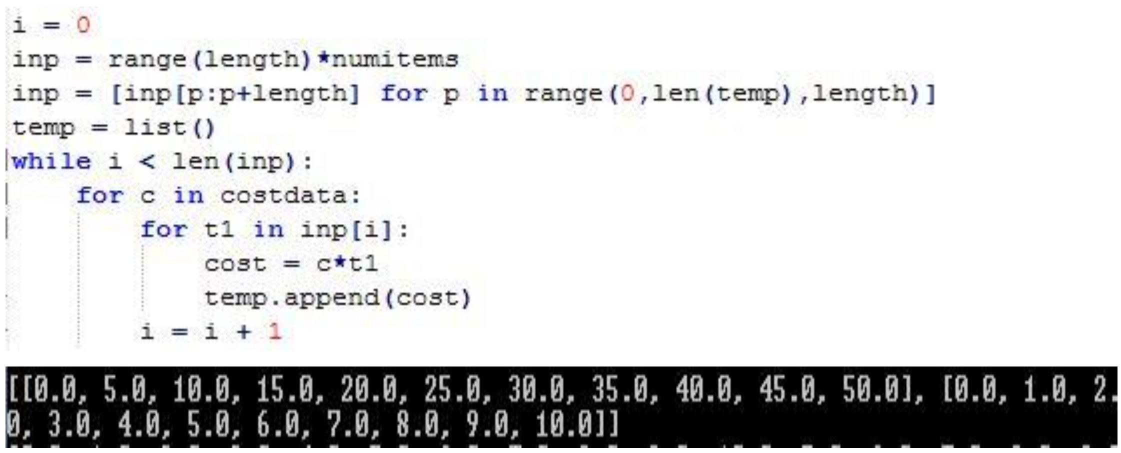


```
Please enter unit cost for item 1: 5
Please enter unit cost for item 2: 1
Please check the unit cost data once again !!!
Type (Yes) in case you want to continue, (No) to reinput the data.
Continue or not ? Yes
[5.0, 1.0]
```

Figure 20. Asking inputs for unit cost of every item

5.4.2 Making a list of total cost of each stock level for every item

The underlying idea to build up this list is using two finite loops (loop in loop) technique. Firstly, a loop starts from the first position from the costdata list (unit cost 5 dollars). Then the second loop will runs in a temporary list of stock levels. The first loop will continue to the second position, only if the second one finish the temporary list. The result is a cost list comprises of sublists for every items. And the total cost for each stock levels are inside that sublists.



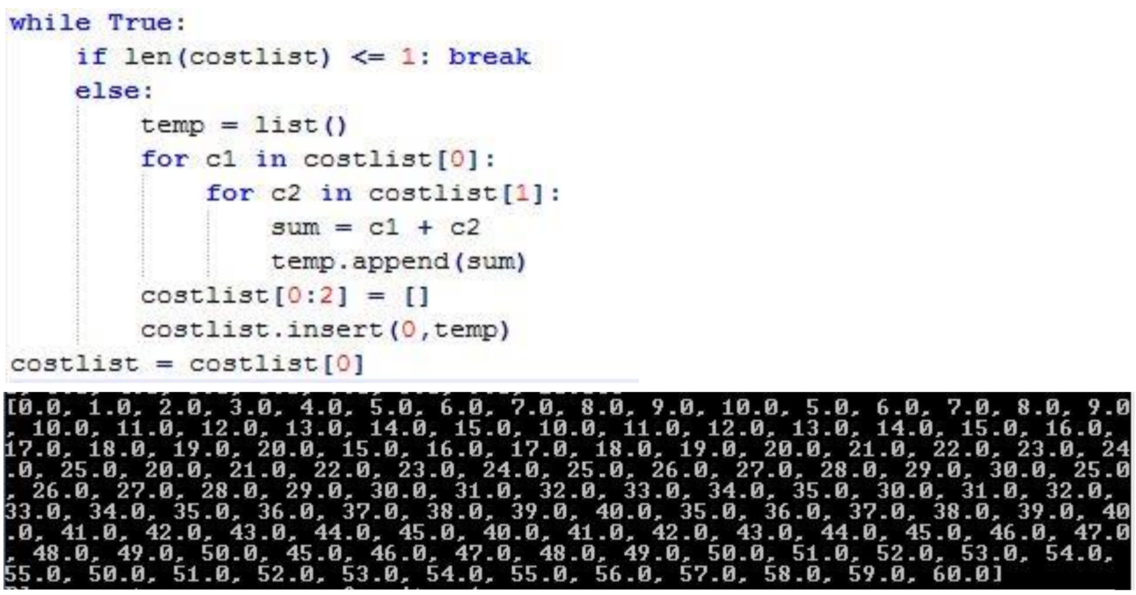
```
i = 0
inp = range(length)*numitems
inp = [inp[p:p+length] for p in range(0,len(temp),length)]
temp = list()
while i < len(inp):
    for c in costdata:
        for t1 in inp[i]:
            cost = c*t1
            temp.append(cost)
        i = i + 1
```

```
[[0.0, 5.0, 10.0, 15.0, 20.0, 25.0, 30.0, 35.0, 40.0, 45.0, 50.0], [0.0, 1.0, 2.
0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0]]
```

Figure 21. Making a total cost list for each stock level of every item

5.4.3 Making final cost list for each combination of stock level

The same technique of two finite loops was used to create a final list which shows the total cost for each combination of stock level. For a unit which has over 2 critical items, a new technique was applied. Firstly, the program will take the sum of the combination of two first items. Then, the data will be added in to a temporary list. After that, the program will calculate the total sum between the temporary list and the third item cost list and so on.



```
while True:
    if len(costlist) <= 1: break
    else:
        temp = list()
        for c1 in costlist[0]:
            for c2 in costlist[1]:
                sum = c1 + c2
                temp.append(sum)
        costlist[0:2] = []
        costlist.insert(0,temp)
costlist = costlist[0]
```

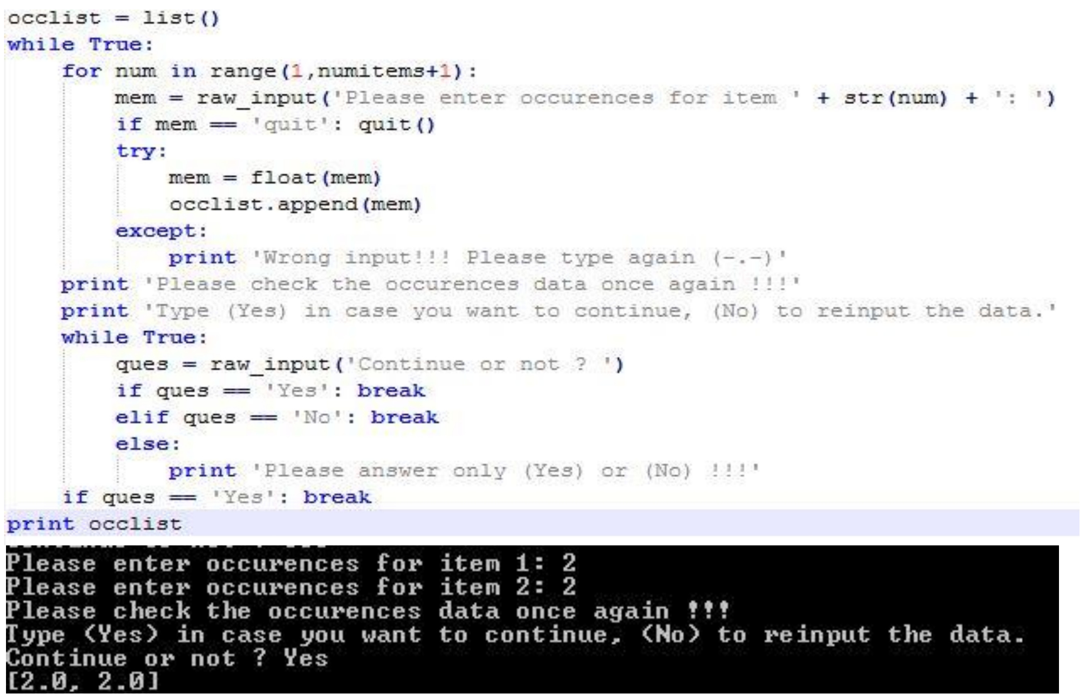
```
[0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 5.0, 6.0, 7.0, 8.0, 9.0
, 10.0, 11.0, 12.0, 13.0, 14.0, 15.0, 10.0, 11.0, 12.0, 13.0, 14.0, 15.0, 16.0,
17.0, 18.0, 19.0, 20.0, 15.0, 16.0, 17.0, 18.0, 19.0, 20.0, 21.0, 22.0, 23.0, 24
.0, 25.0, 20.0, 21.0, 22.0, 23.0, 24.0, 25.0, 26.0, 27.0, 28.0, 29.0, 30.0, 25.0
, 26.0, 27.0, 28.0, 29.0, 30.0, 31.0, 32.0, 33.0, 34.0, 35.0, 30.0, 31.0, 32.0,
33.0, 34.0, 35.0, 36.0, 37.0, 38.0, 39.0, 40.0, 35.0, 36.0, 37.0, 38.0, 39.0, 40
.0, 41.0, 42.0, 43.0, 44.0, 45.0, 40.0, 41.0, 42.0, 43.0, 44.0, 45.0, 46.0, 47.0
, 48.0, 49.0, 50.0, 45.0, 46.0, 47.0, 48.0, 49.0, 50.0, 51.0, 52.0, 53.0, 54.0,
55.0, 50.0, 51.0, 52.0, 53.0, 54.0, 55.0, 56.0, 57.0, 58.0, 59.0, 60.0]
```

Figure 22. Making final cost list for each combination of stock level

5.5 Step 5 - Creating a list for system availability

5.5.1 Making a list for occurences

Beforehand, an empty list was created and named ”occlist”. By using infinite and finite loop, a list of occurrences for every critical items was constructed. The infinite loop in this case just helps user to check the input/result and edit the data again if needed. The result is like this.



```
occlist = list()
while True:
    for num in range(1,numitems+1):
        mem = raw_input('Please enter occurences for item ' + str(num) + ': ')
        if mem == 'quit': quit()
        try:
            mem = float(mem)
            occlist.append(mem)
        except:
            print 'Wrong input!!! Please type again (-.-)'
    print 'Please check the occurences data once again !!!'
    print 'Type (Yes) in case you want to continue, (No) to reinput the data.'
    while True:
        ques = raw_input('Continue or not ? ')
        if ques == 'Yes': break
        elif ques == 'No': break
        else:
            print 'Please answer only (Yes) or (No) !!!'
    if ques == 'Yes': break
print occlist
```

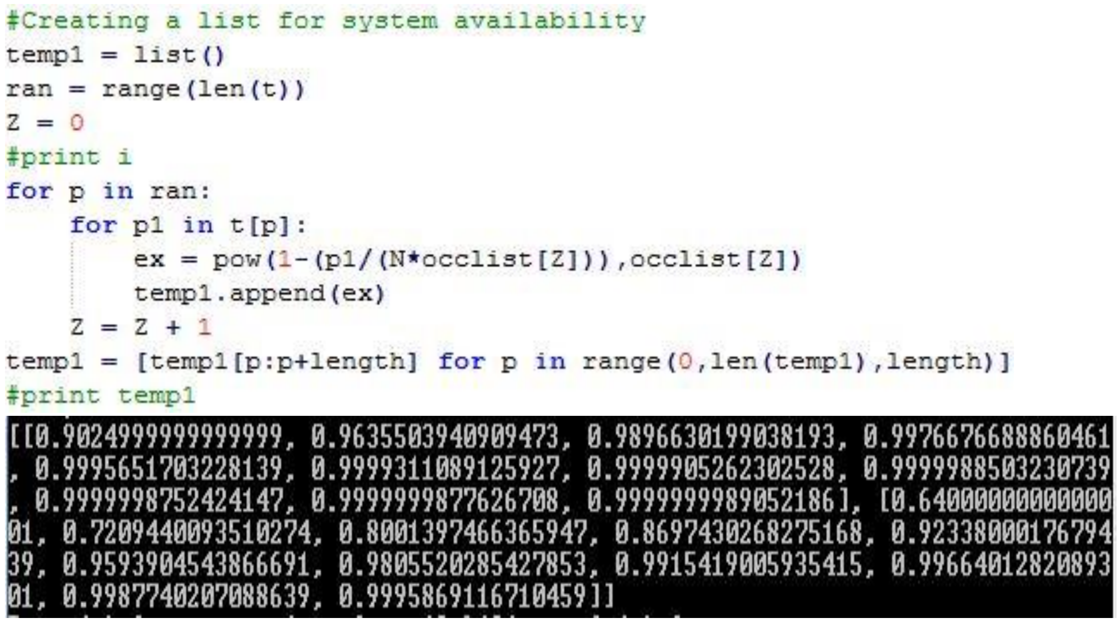
```
Please enter occurences for item 1: 2
Please enter occurences for item 2: 2
Please check the occurences data once again !!!
Type (Yes) in case you want to continue, (No) to reinput the data.
Continue or not ? Yes
[2.0, 2.0]
```

Figure 23. Making a list for occurrences

5.5.2 Calculating availability of each item for every stock level

The system availability is calculated by using the formula in section 3.10 (Adapted from Sherbrooke 2004, 39).

At this step, the calculation for every availability is computed first based on these variables: EBO, fleet size (N), occurrences Z. Then it is added to a temporary list called ”temp1”.



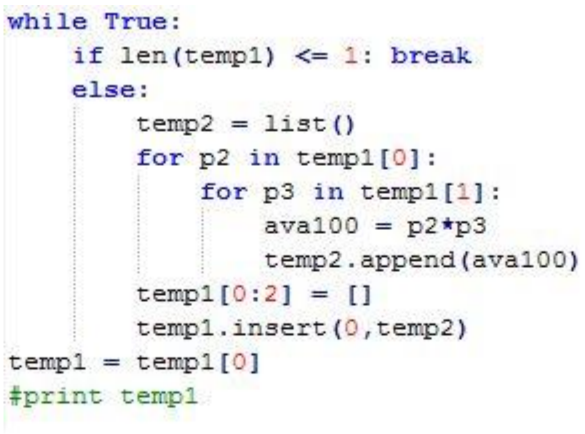
```
#Creating a list for system availability
temp1 = list()
ran = range(len(t))
Z = 0
#print i
for p in ran:
    for p1 in t[p]:
        ex = pow(1-(p1/(N*occlist[Z])),occlist[Z])
        temp1.append(ex)
    Z = Z + 1
temp1 = [temp1[p:p+length] for p in range(0,len(temp1),length)]
#print temp1
```

```
[[0.9024999999999999, 0.9635503940909473, 0.9896630199038193, 0.9976676688860461
, 0.9995651703228139, 0.9999311089125927, 0.9999905262302528, 0.9999988503230739
, 0.9999998752424147, 0.9999999877626708, 0.9999999989052186], [0.64000000000000
01, 0.7209440093510274, 0.8001397466365947, 0.8697430268275168, 0.92338000176794
39, 0.9593904543866691, 0.9805520285427853, 0.9915419005935415, 0.99664012820893
01, 0.9987740207088639, 0.9995869116710459]]
```

Figure 24. Making an availability list of each stock level for every item

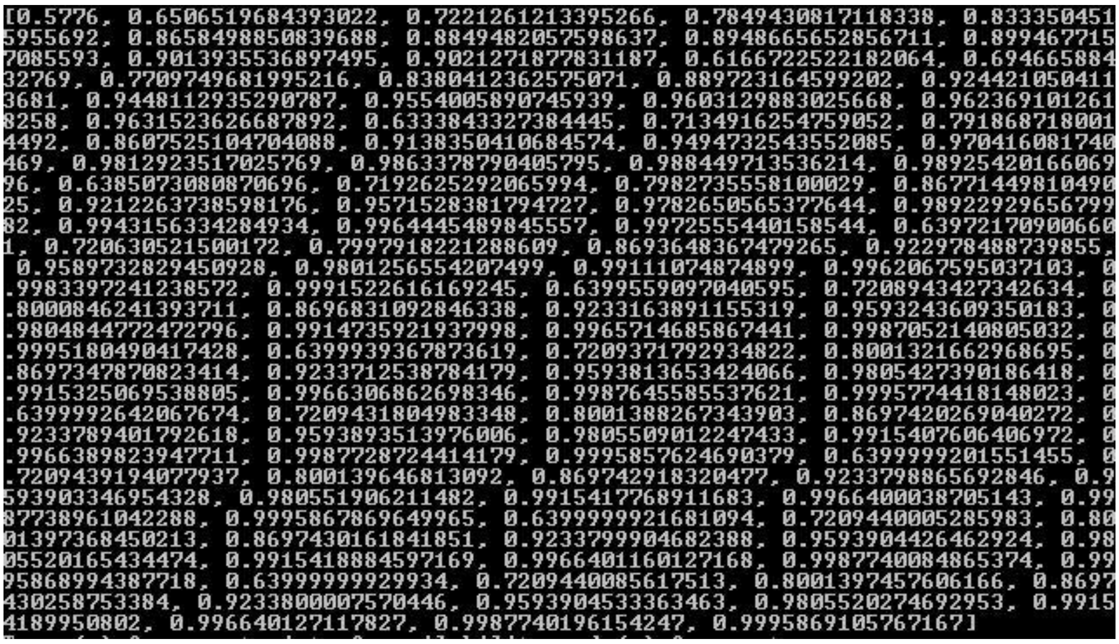
5.5.3 Calculating and making a list for system availability

The system availability is the product of availability of every items at certain stock level. The result is a list of system availability for each combination of stock level.



```python
while True:
    if len(temp1) <= 1: break
    else:
        temp2 = list()
        for p2 in temp1[0]:
            for p3 in temp1[1]:
                ava100 = p2*p3
                temp2.append(ava100)
        temp1[0:2] = []
        temp1.insert(0,temp2)
temp1 = temp1[0]
#print temp1
```

Figure 25. System availability calculation syntax



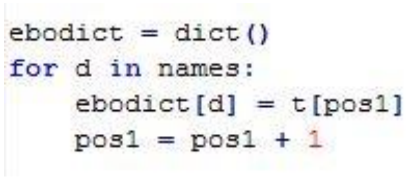
```
[0.5776, 0.6506519684393022, 0.7221261213395266, 0.7849430817118338, 0.833350451
5955692, 0.8658498850839688, 0.8849482057598637, 0.8948665652856711, 0.899467715
7085593, 0.9013935536897495, 0.9021271877831187, 0.6166722522182064, 0.694665884
32769, 0.7709749681995216, 0.8380412362575071, 0.889723164599202, 0.924421050411
3681, 0.9448112935290787, 0.9554005890745939, 0.9603129883025668, 0.962369101261
8258, 0.9631523626687892, 0.6333843327384445, 0.7134916254759052, 0.791868718001
4492, 0.8607525104704088, 0.9138350410684574, 0.9494732543552085, 0.970416081740
469, 0.9812923517025769, 0.9863378790405795, 0.988449713536214, 0.9892542016606
96, 0.6385073080870696, 0.7192625292065994, 0.7982735558100029, 0.8677144981049
25, 0.9212263738598176, 0.9571528381794727, 0.9782650565377644, 0.9892292965679
82, 0.9943156334284934, 0.9964445489845557, 0.9972555440158544, 0.6397217090066
1, 0.720630521500172, 0.7997918221288609, 0.8693648367479265, 0.922978488739855,
 0.9589732829450928, 0.9801256554207499, 0.99111074874899, 0.9962067595037103, 0
.9983397241238572, 0.9991522616169245, 0.6399559097040595, 0.7208943427342634, 0
.8000846241393711, 0.8696831092846338, 0.9233163891155319, 0.9593243609350183, 0
.9804844772472796, 0.9914735921937998, 0.9965714685867441, 0.9987052140805032, 0
.9995180490417428, 0.6399939367873619, 0.7209371792934822, 0.8001321662968695, 0
.8697347870823414, 0.9233712538784179, 0.9593813653424066, 0.9805427390186418, 0
.9915325069538805, 0.9966306862698346, 0.9987645585537621, 0.9995774418148023, 0
.6399992642067674, 0.7209431804983348, 0.8001388267343903, 0.8697420269040272, 0
.9233789401792618, 0.9593893513976006, 0.9805509012247433, 0.9915407606406972, 0
.9966389823947711, 0.9987728724414179, 0.9995857624690379, 0.6399999201551455, 0
.7209439194077937, 0.8001396468130092, 0.869742918320477, 0.9233798865692846, 0.9
593903346954328, 0.980551906211482, 0.9915417768911683, 0.9966400038705143, 0.99
87738961042288, 0.9995867869649965, 0.6399999921681094, 0.7209440005285983, 0.80
01397368450213, 0.8697430161841851, 0.9233799904682388, 0.9593904426462924, 0.98
05520165434474, 0.9915418884597169, 0.9966401160127168, 0.9987740084865374, 0.99
95868994387718, 0.6399999999929934, 0.7209440085617513, 0.8001397457606166, 0.8697
430258753384, 0.9233800007570446, 0.9593904533363463, 0.9805520274692953, 0.9915
4189950802, 0.9966401271178270, 0.9987740196154247, 0.9995869105767167]
```

Figure 26. A List of system availability for all possible combinations

5.6 Step 6 - Creating dictionaries

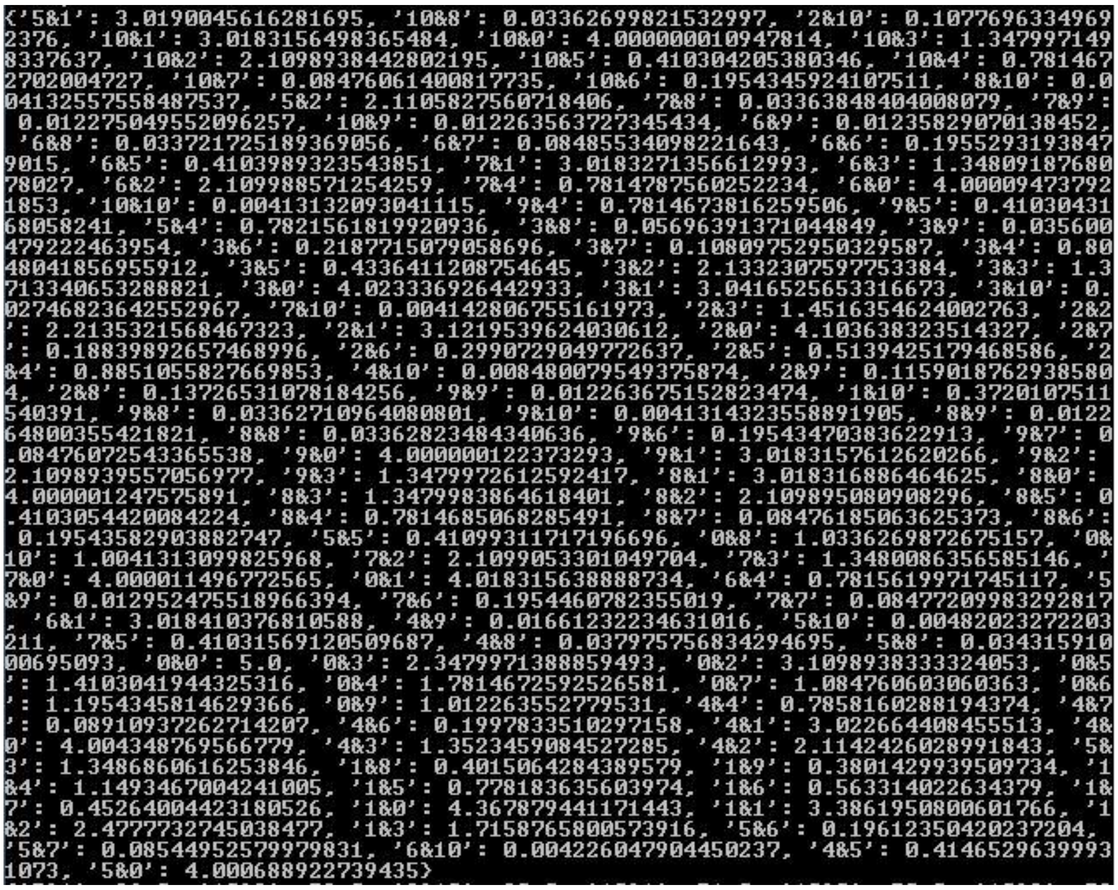
5.6.1 Dictionary of EBOs

The dictionary of EBOs includes keys and values. The keys are the combination of different stock level from item 1 and 2 in this example. The simultaneous value are expected back order value from EBO list ”t”.



```
ebodict = dict()
for d in names:
    ebodict[d] = t[pos1]
    pos1 = pos1 + 1
```

Figure 27. Dictionary of EBOs syntax

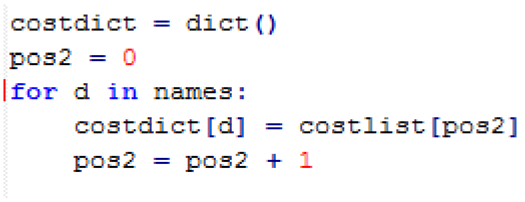


```
{'5&1': 3.0190045616281695, '10&8': 0.03362699821532997, '2&10': 0.1077696334969
2376, '10&1': 3.0183156498365484, '10&0': 4.000000010947814, '10&3': 1.347997149
8337637, '10&2': 2.1098938442802195, '10&5': 0.410304205380346, '10&4': 0.781467
2702004727, '10&7': 0.08476061400817735, '10&6': 0.1954345924107511, '8&10': 0.0
0413255755848753 7, '5&2': 2.1105827560718406, '7&8': 0.03363848404008079, '7&9':
 0.012275049552096257, '10&9': 0.012263563727345434, '6&9': 0.012358290701384 52,
 '6&8': 0.03372172518936905 6, '6&7': 0.08485534098221643, '6&6': 0.1955293193847
9015, '6&5': 0.4103989323543851, '7&1': 3.0183271356612993, '6&3': 1.34809187680
78027, '6&2': 2.109988571254259, '7&4': 0.7814787560252234, '6&0': 4.00009473792
1853, '10&10': 0.004131320930411115, '9&4': 0.7814673816259506, '9&5': 0.41030431
68058241, '5&4': 0.7821561819920936, '3&8': 0.05696391371044849, '3&9': 0.035600
47922246395 4, '3&6': 0.2187715079058696, '3&7': 0.10809752950329587, '3&4': 0.80
48041856955912, '3&5': 0.4336411208754645, '3&2': 2.1332307597753384, '3&3': 1.3
713340653288821, '3&0': 4.023336926442933, '3&1': 3.0416525653316673, '3&10': 0.
02746823642552967, '7&10': 0.004142806755161973, '2&3': 1.4516354624002763, '2&2
': 2.2135321568467323, '2&1': 3.1219539624030612, '2&0': 4.103638323514327, '2&7
': 0.18839892657468996, '2&6': 0.2990729049772637, '2&5': 0.5139425179468586, '2
&4': 0.8851055827669853, '4&10': 0.008480079549375874, '2&9': 0.1159018762938580
4, '2&8': 0.13726531078184256, '9&9': 0.012263675152823474, '1&10': 0.3720107511
540391, '9&8': 0.03362710964080801, '9&10': 0.004131432355889190 5, '8&9': 0.0122
64800355421821, '8&8': 0.03362823484340636, '9&6': 0.19543470383622913, '9&7': 0
.08476072543365538, '9&0': 4.000000122373293, '9&1': 3.0183157612620266, '9&2':
2.1098939557056977, '9&3': 1.3479972612592417, '8&1': 3.018316886464625, '8&0':
4.000001247575891, '8&3': 1.3479983864618401, '8&2': 2.109895080908296, '8&5': 0
.4103054420084224, '8&4': 0.7814685068285491, '8&7': 0.08476185063625373, '8&6':
 0.19543582903882747, '5&5': 0.4109931171719669 6, '0&8': 1.0336269872675157, '0&
10': 1.0041313099825968, '7&2': 2.1099053301049704, '7&3': 1.3480086356585146, '
7&0': 4.000011496772565, '0&1': 4.018315638888734, '6&4': 0.7815619971745117, '5
&9': 0.012952475518966394, '7&6': 0.195446078235501 9, '7&7': 0.0847720998329281 7
, '6&1': 3.018410376810588, '4&9': 0.01661232234631016, '5&10': 0.0048202327220 3
211, '7&5': 0.41031569120509687, '4&8': 0.03797575683429469 5, '5&8': 0.03431591 0
00695093, '0&0': 5.0, '0&3': 2.347997138885949 3, '0&2': 3.10989383332405 3, '0&5
': 1.4103041944325316, '0&4': 1.7814672592526581, '0&7': 1.084760603060363, '0&6
': 1.195434581462936 6, '0&9': 1.012263552779531, '4&4': 0.7858160288194374, '4&7
': 0.08910937262714207, '4&6': 0.1997833510297158, '4&1': 3.022664408455513, '4&
0': 4.004348769566779, '4&3': 1.3523459084527285, '4&2': 2.1142426028991843, '5&
3': 1.3486860616253846, '1&8': 0.4015064284389579, '1&9': 0.3801429939509734, '1
&4': 1.1493467004241005, '1&5': 0.778183635603974, '1&6': 0.5633140226343 79, '1&
7': 0.4526400442318052 6, '1&0': 4.367879441171443, '1&1': 3.386195080060176 6, '1
&2': 2.4777327450384 77, '1&3': 1.715876580057391 6, '5&6': 0.19612350420237204,
'5&7': 0.0854495257997983 1, '6&10': 0.004226047904450237, '4&5': 0.4146529639993
1073, '5&0': 4.000688922739435}
```

Figure 28. Dictionary of EBOs

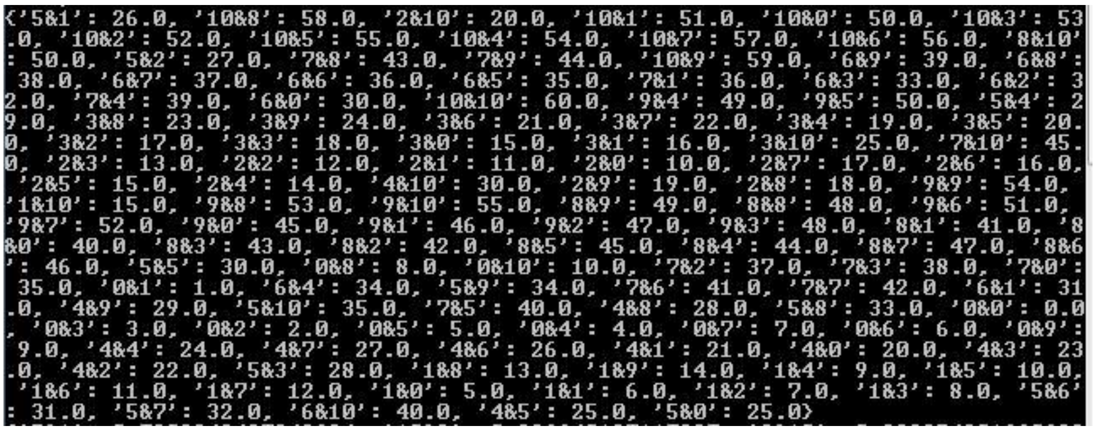
5.6.2 Dictionary of costlist

For cost list the code is pretty the same. It will give information about the name of certain combination of stock level and the total cost for keeping those spares.



```
costdict = dict()
pos2 = 0
for d in names:
    costdict[d] = costlist[pos2]
    pos2 = pos2 + 1
```

Figure 29. Dictionary of costlist syntax

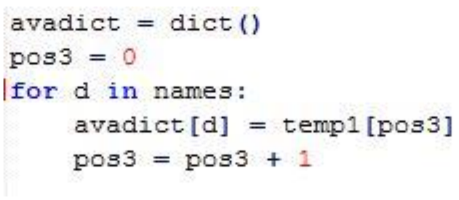


```
{'5&1': 26.0, '10&8': 58.0, '2&10': 20.0, '10&1': 51.0, '10&0': 50.0, '10&3': 53
.0, '10&2': 52.0, '10&5': 55.0, '10&4': 54.0, '10&7': 57.0, '10&6': 56.0, '8&10'
: 50.0, '5&2': 27.0, '7&8': 43.0, '7&9': 44.0, '10&9': 59.0, '6&9': 39.0, '6&8':
 38.0, '6&7': 37.0, '6&6': 36.0, '6&5': 35.0, '7&1': 36.0, '6&3': 33.0, '6&2': 3
2.0, '7&4': 39.0, '6&0': 30.0, '10&10': 60.0, '9&4': 49.0, '9&5': 50.0, '5&4': 2
9.0, '3&8': 23.0, '3&9': 24.0, '3&6': 21.0, '3&7': 22.0, '3&4': 19.0, '3&5': 20.
0, '3&2': 17.0, '3&3': 18.0, '3&0': 15.0, '3&1': 16.0, '3&10': 25.0, '7&10': 45.
0, '2&3': 13.0, '2&2': 12.0, '2&1': 11.0, '2&0': 10.0, '2&7': 17.0, '2&6': 16.0,
 '2&5': 15.0, '2&4': 14.0, '4&10': 30.0, '2&9': 19.0, '2&8': 18.0, '9&9': 54.0,
'1&10': 15.0, '9&8': 53.0, '9&10': 55.0, '8&9': 49.0, '8&8': 48.0, '9&6': 51.0,
'9&7': 52.0, '9&0': 45.0, '9&1': 46.0, '9&2': 47.0, '9&3': 48.0, '8&1': 41.0, '8
&0': 40.0, '8&3': 43.0, '8&2': 42.0, '8&5': 45.0, '8&4': 44.0, '8&7': 47.0, '8&6
': 46.0, '5&5': 30.0, '0&8': 8.0, '0&10': 10.0, '7&2': 37.0, '7&3': 38.0, '7&0':
 35.0, '0&1': 1.0, '6&4': 34.0, '5&9': 34.0, '7&6': 41.0, '7&7': 42.0, '6&1': 31
.0, '4&9': 29.0, '5&10': 35.0, '7&5': 40.0, '4&8': 28.0, '5&8': 33.0, '0&0': 0.0
, '0&3': 3.0, '0&2': 2.0, '0&5': 5.0, '0&4': 4.0, '0&7': 7.0, '0&6': 6.0, '0&9':
 9.0, '4&4': 24.0, '4&7': 27.0, '4&6': 26.0, '4&1': 21.0, '4&0': 20.0, '4&3': 23
.0, '4&2': 22.0, '5&3': 28.0, '1&8': 13.0, '1&9': 14.0, '1&4': 9.0, '1&5': 10.0,
 '1&6': 11.0, '1&7': 12.0, '1&0': 5.0, '1&1': 6.0, '1&2': 7.0, '1&3': 8.0, '5&6'
: 31.0, '5&7': 32.0, '6&10': 40.0, '4&5': 25.0, '5&0': 25.0}
```

Figure 30. Dictionary of costlist

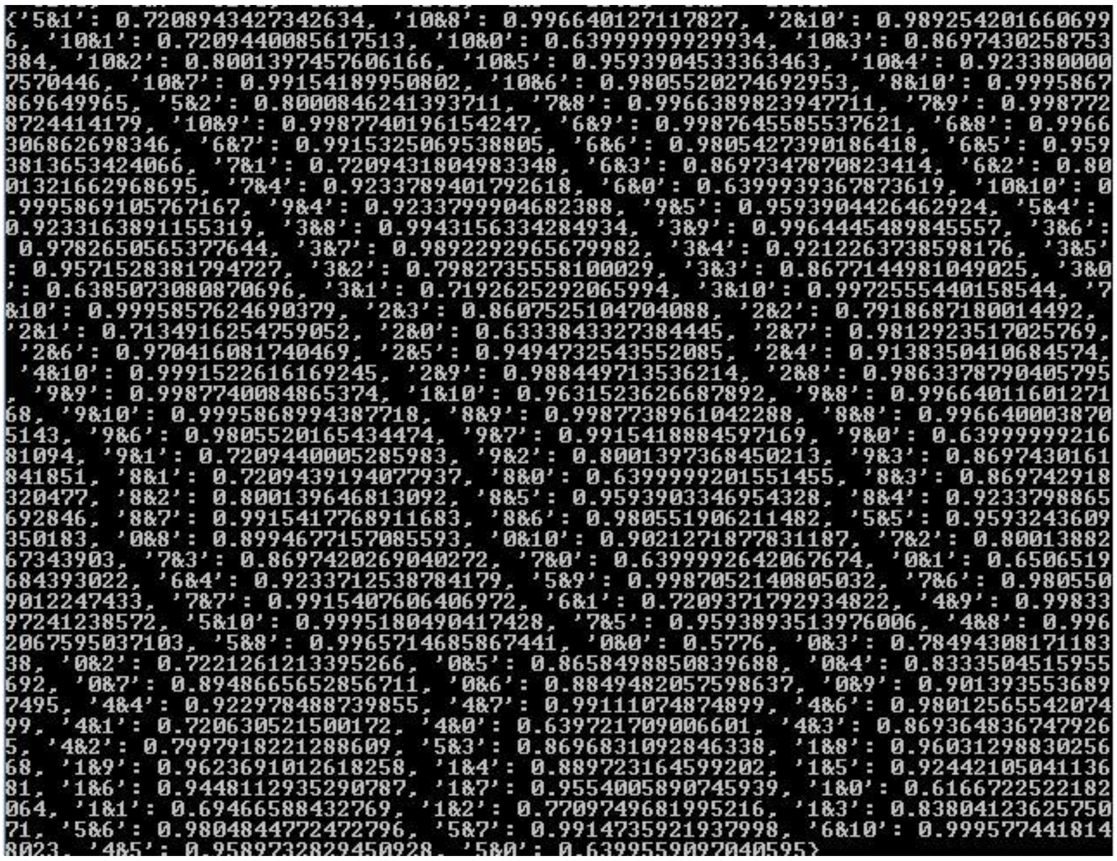
5.6.3 Dictionary of system availability

By applying the similar code, a dictionary was created which includes key name for combination of stock level and the system availability for each option.



```
avadict = dict()
pos3 = 0
for d in names:
    avadict[d] = temp1[pos3]
    pos3 = pos3 + 1
```

Figure 31. Dictionary of system availability syntax



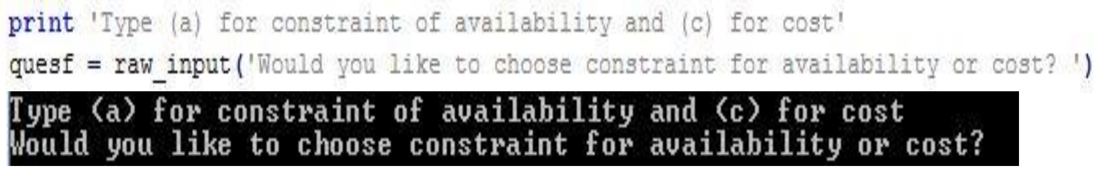
```
{'5&1': 0.7208943427342634, '10&8': 0.996640127117827, '2&10': 0.989254201660699
6, '10&1': 0.7209440085617513, '10&0': 0.63999999929934, '10&3': 0.8697430258753
384, '10&2': 0.8001397457606166, '10&5': 0.9593904533363463, '10&4': 0.923380000
7570446, '10&7': 0.99154189950802, '10&6': 0.980552027469295З, '8&10': 0.9995867
869649965, '5&2': 0.8000846241393711, '7&8': 0.9966389823947711, '7&9': 0.998772
8724414179, '10&9': 0.9987740196154247, '6&9': 0.9987645585537621, '6&8': 0.9966
306862698346, '6&7': 0.9915325069538805, '6&6': 0.9805427390186418, '6&5': 0.959
3813653424066, '7&1': 0.7209431804983348, '6&3': 0.8697347870823414, '6&2': 0.80
01321662968695, '7&4': 0.9233789401792618, '6&0': 0.6399939367873619, '10&10': 0
.9995869105767167, '9&4': 0.9233799904682388, '9&5': 0.9593904426462924, '5&4':
0.9233163891155319, '3&8': 0.9943156334284934, '3&9': 0.9964445489845557, '3&6':
 0.9782650565377644, '3&7': 0.9892292965679982, '3&4': 0.9212263738598176, '3&5'
: 0.9571528381794727, '3&2': 0.7982735558100029, '3&3': 0.8677144981049025, '3&0
': 0.6385073080870696, '3&1': 0.7192625292065994, '3&10': 0.9972555440158544, '7
&10': 0.9995857624690379, '2&3': 0.8607525104704088, '2&2': 0.7918687180014492,
'2&1': 0.7134916254759052, '2&0': 0.6333843327384445, '2&7': 0.9812923517025769,
 '2&6': 0.970416081740469, '2&5': 0.9494732543552085, '2&4': 0.9138350410684574,
 '4&10': 0.9991522616169245, '2&9': 0.988449713536214, '2&8': 0.9863378790405795
, '9&9': 0.9987740084865374, '1&10': 0.9631523626687892, '9&8': 0.99664011601271
68, '9&10': 0.9995868994387718, '8&9': 0.9987738961042288, '8&8': 0.996640003870
5143, '9&6': 0.9805520165434474, '9&7': 0.9915418884597169, '9&0': 0.63999999216
81094, '9&1': 0.7209440005285983, '9&2': 0.8001397368450213, '9&3': 0.8697430161
841851, '8&1': 0.7209439194077937, '8&0': 0.6399999201551455, '8&3': 0.869742918
320477, '8&2': 0.800139646813092, '8&5': 0.9593903346954328, '8&4': 0.9233798865
692846, '8&7': 0.9915417768911683, '8&6': 0.980551906211482, '5&5': 0.9593243609
350183, '0&8': 0.8994677157085593, '0&10': 0.9021271877831187, '7&2': 0.80013882
67343903, '7&3': 0.8697420269040272, '7&0': 0.6399992642067674, '0&1': 0.6506519
684393022, '6&4': 0.9233712538784179, '5&9': 0.9987052140805032, '7&6': 0.980550
9012247433, '7&7': 0.9915407606406972, '6&1': 0.7209371792934822, '4&9': 0.99833
97241238572, '5&10': 0.9995180490417428, '7&5': 0.9593893513976006, '4&8': 0.996
2067595037103, '5&8': 0.9965714685867441, '0&0': 0.5776, '0&3': 0.78494308171183
38, '0&2': 0.7221261213395266, '0&5': 0.8658498850839688, '0&4': 0.833350451595
692, '0&7': 0.894866565285671l, '0&6': 0.8849482057598637, '0&9': 0.901393553689
7495, '4&4': 0.922978488739855, '4&7': 0.9911107487489, '4&6': 0.980125655542074
99, '4&1': 0.720630521500172, '4&0': 0.639721709006601, '4&3': 0.869364836747926
5, '4&2': 0.7997918221288609, '5&3': 0.8696831092846338, '1&8': 0.96031298830256
68, '1&9': 0.9623691012618258, '1&4': 0.889723164599202, '1&5': 0.92442105041136
81, '1&6': 0.9448112935290787, '1&7': 0.9554005890745939, '1&0': 0.61667225221 82
064, '1&1': 0.69466588432769, '1&2': 0.7709749681995216, '1&3': 0.83804123625750
71, '5&6': 0.9804844772472796, '5&7': 0.9914735921937998, '6&10': 0.999577441814
8023, '4&5': 0.9589732829450928, '5&0': 0.6399559097040595}
```

Figure 32. Dictionary of system availability

5.7 Step 7 – Finding optimal inventory policy

5.7.1 Designing for type of constraints

In this model, there are two types of constraint. The former is availability constraint and the latter is inventory cost constraint. So a question was made in order to ask the user to choose between one of them.



```
print 'Type (a) for constraint of availability and (c) for cost'
quesf = raw_input('Would you like to choose constraint for availability or cost? ')
```

```
Type (a) for constraint of availability and (c) for cost
Would you like to choose constraint for availability or cost?
```

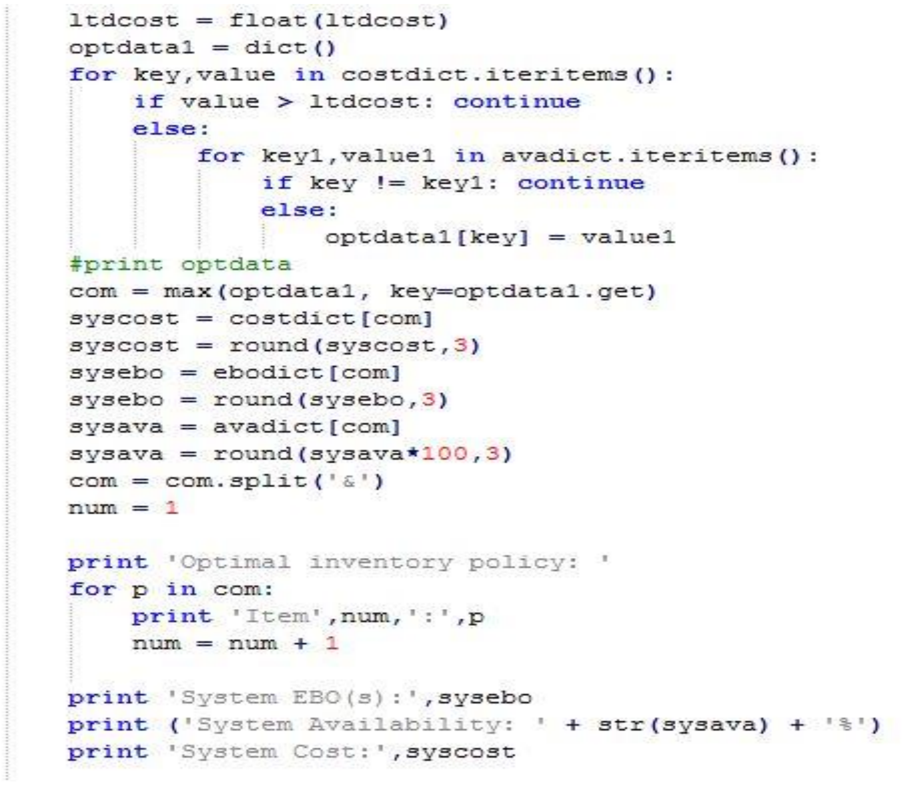
Figure 33. Asking for constraint syntax

5.7.2 Optimizing inventory policy with inventory cost constraint

The priority condition to run the optimization is that the limited cost which is entered by the user has to be higher, then the minimum total cost for all combination of stock level of items. If user enters wrong numbers, the program will give a warning and then force the user to input again the limited cost.

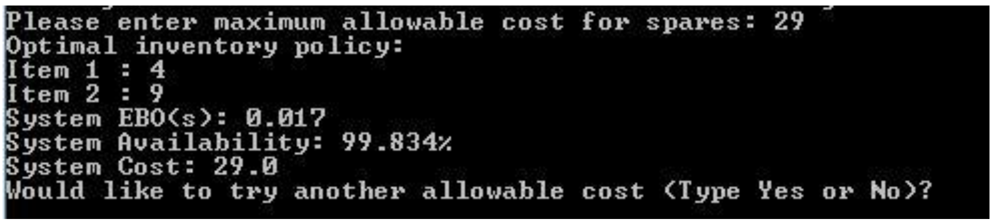
In case of valid input, the program with compare the cost constraint with each cost in the dictionary of cost. If the cost in the dictionary is lower than limited cost, the program will remember the key names.

Based on the key names above, the program continues to look up in the dictionary of system availability to choose one key name with highest system availability.



```
ltdcost = float(ltdcost)
optdata1 = dict()
for key,value in costdict.iteritems():
    if value > ltdcost: continue
    else:
        for key1,value1 in avadict.iteritems():
            if key != key1: continue
            else:
                optdata1[key] = value1
#print optdata
com = max(optdata1, key=optdata1.get)
syscost = costdict[com]
syscost = round(syscost,3)
sysebo = ebodict[com]
sysebo = round(sysebo,3)
sysava = avadict[com]
sysava = round(sysava*100,3)
com = com.split('&')
num = 1

print 'Optimal inventory policy: '
for p in com:
    print 'Item',num,':',p
    num = num + 1

print 'System EBO(s):',sysebo
print ('System Availability: ' + str(sysava) + '%')
print 'System Cost:',syscost
```

Figure 34. Optimization of inventory policy with inventory cost constraint syntax



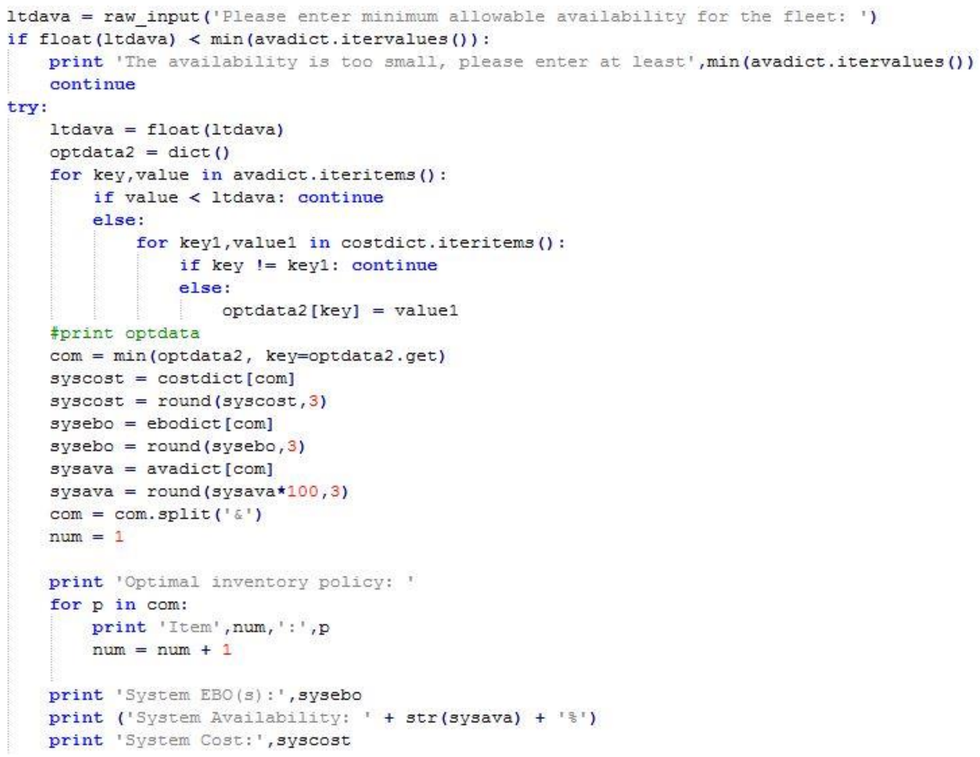
```
Please enter maximum allowable cost for spares: 29
Optimal inventory policy:
Item 1 : 4
Item 2 : 9
System EBO(s): 0.017
System Availability: 99.834%
System Cost: 29.0
Would like to try another allowable cost (Type Yes or No)?
```

Figure 35. Optimal stock level result with inventory cost constraint

5.7.3 Optimizing inventory policy with system availability constraint

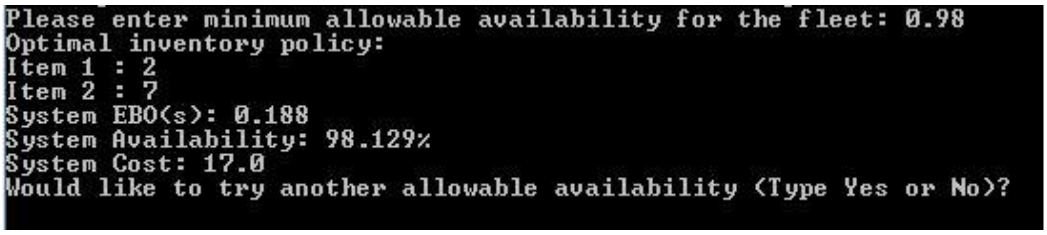
According to availability constraint, the process is in the opposite direction. Because availability is a floor constraint (minimum), it has to be greater than the maximum constraint in the dictionary of availability.

If the input is valid, the program will take that constraint in to comparison with each value in the dictionary and only remember key names which has higher availability then the constraint. After that, the program will find the only key with minimum cost within those key names.



```
ltdava = raw_input('Please enter minimum allowable availability for the fleet: ')
if float(ltdava) < min(avadict.itervalues()):
    print 'The availability is too small, please enter at least',min(avadict.itervalues())
    continue
try:
    ltdava = float(ltdava)
    optdata2 = dict()
    for key,value in avadict.iteritems():
        if value < ltdava: continue
        else:
            for key1,value1 in costdict.iteritems():
                if key != key1: continue
                else:
                    optdata2[key] = value1
    #print optdata
    com = min(optdata2, key=optdata2.get)
    syscost = costdict[com]
    syscost = round(syscost,3)
    sysebo = ebodict[com]
    sysebo = round(sysebo,3)
    sysava = avadict[com]
    sysava = round(sysava*100,3)
    com = com.split('&')
    num = 1

    print 'Optimal inventory policy: '
    for p in com:
        print 'Item',num,':',p
        num = num + 1

    print 'System EBO(s):',sysebo
    print ('System Availability: ' + str(sysava) + '%')
    print 'System Cost:',syscost
```

Figure 36. Optimizing inventory policy with system availability constraint

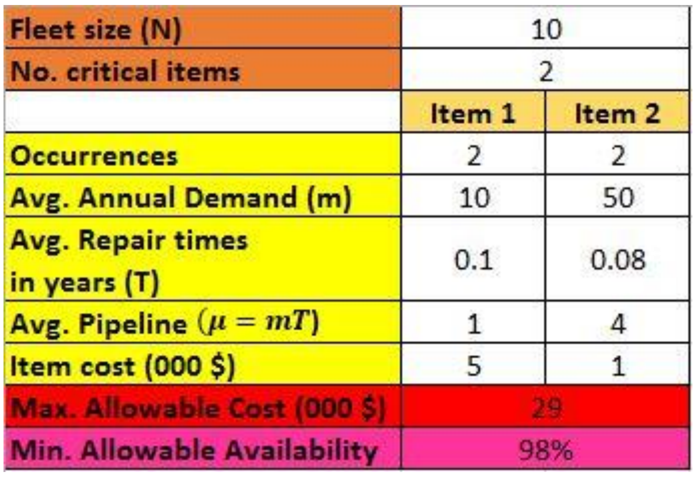


```
Please enter minimum allowable availability for the fleet: 0.98
Optimal inventory policy:
Item 1 : 2
Item 2 : 7
System EBO(s): 0.188
System Availability: 98.129%
System Cost: 17.0
Would like to try another allowable availability (Type Yes or No)?
```

Figure 37. Optimal stock level result with system availability constraint

6 RESULTS

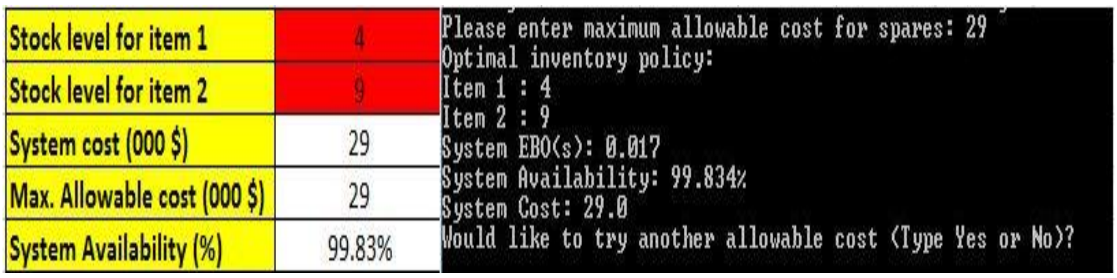
By using two different ways to optimize the number of spare units for each item, a comparison for the result was needed, in order to validate the program constructed in Python. For a test, a list of same inputs was put into the model in Excel and Python. Those inputs are illustrated in a table below.



Fleet size (N)	10	
No. critical items	2	
	Item 1	Item 2
Occurrences	2	2
Avg. Annual Demand (m)	10	50
Avg. Repair times in years (T)	0.1	0.08
Avg. Pipeline ($\mu = mT$)	1	4
Item cost (000 $)	5	1
Max. Allowable Cost (000 $)	29	
Min. Allowable Availability	98%	

Figure 38. Sample data inputs

There were two comparisons corresponding to two types of constraint. Users should notice that only one constraint (either maximum cost or minimum availability) is chosen for the optimization. With reference to maximum allowable cost constraint which was set to 29,000 $, both techniques gave the same result. The optimal spares for item 1 is 4 and item 2 is 9. In addition, the availability of the system was 99.83% and system cost was 29,000 dollars.

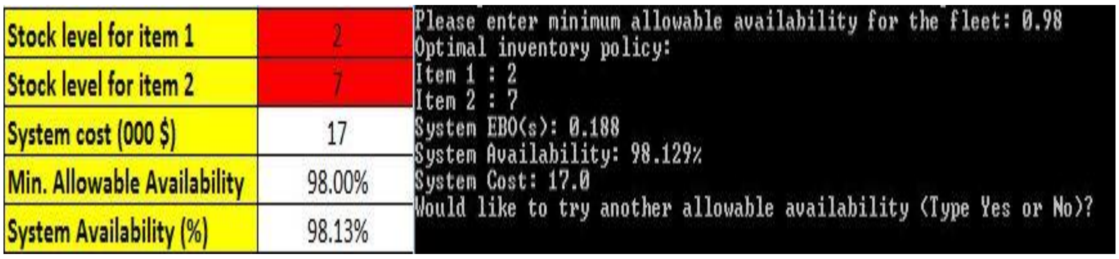


Stock level for item 1	4
Stock level for item 2	9
System cost (000 $)	29
Max. Allowable cost (000 $)	29
System Availability (%)	99.83%

```
Please enter maximum allowable cost for spares: 29
Optimal inventory policy:
Item 1 : 4
Item 2 : 9
System EBO(s): 0.017
System Availability: 99.834%
System Cost: 29.0
Would like to try another allowable cost (Type Yes or No)?
```

Figure 39. Compared results using two different techniques with inventory cost constraint

According to availability constraint (minimum 98%), the results remained the same again. The optimal spares for item 1 is 2 and item 2 is 7. The availability of the system and system cost were 98.13% and 17,000 dollars, respectively.



Stock level for item 1	2
Stock level for item 2	7
System cost (000 $)	17
Min. Allowable Availability	98.00%
System Availability (%)	98.13%

```
Please enter minimum allowable availability for the fleet: 0.98
Optimal inventory policy:
Item 1 : 2
Item 2 : 7
System EBO(s): 0.188
System Availability: 98.129%
System Cost: 17.0
Would like to try another allowable availability (Type Yes or No)?
```

Figure 40. Compared results using two different techniques with system availability constraint

7 DISCUSSION

7.1 CONCLUSION

This thesis clarified the process to create the model in Excel corresponding to the model introduced in Craig C. Sherbrooke (2004). Furthermore, the author went beyond that to make the model more applicable in the real life by making a program in Python. The program was much more flexible than Excel because users can easily change the data input whatever they want without making another model like in Excel. The program is not fully considered as a software because it needs GUI (Graphical User Interface), instead, it can only run on the Command Prompt Window with pre-installed Python 2.7.

In conclusion, Craig C. Sherbrooke model to manage repairable-item inventory worked to increase the availability and decrease the inventory cost. Because there was not enough information about the cost to place an order and the annual holding cost rate, a precise comparison between Craig C. Sherbrooke single-site model and EOQ model could not be conducted. However, the algorithm in Python program will choose the optimal option for the stock level of each critical item among all of possible options. It means any other combination will lead to either a lower in availability or a higher inventory cost.

7.2 LIMITATIONS AND SUGGESTIONS FOR FURTHER RESEARCH

Although the algorithm in the program was creative, it could not work when the considered stock level and the number of critical items was really high. For example, when trying a system of 22 critical items and 25 considered stock level, the program showed a failure notice. The reason behind that was the limitation of the size of a list in Python. A list in Python can contain at maximum 536,870,912 elements for 32-bit system. However, there were approximately $25^{22} = 5.68 \times 10^{30}$ elements in a list which exceeded the limitation.

At the time of writing this thesis, the author had just learnt Python for two and a half month without prior knowledge about programming. That is why the source code was written in a complicated way and the program cannot execute for any of data inputs.

For further development of the program, a different approach in writing the code is to apply the marginal analysis technique. Another solution is to use the class-object function in Python which performs different algorithm for storing data. For whom who would like to use Excel to build the model, but want to make it more flexible, Excel Visual Basic programming language is one option.

References

Arnold, J.R.Tony, Chapman, S.N. & Clive, L.M. (2012). **Introduction to Materials Management**. 7th edition. Pearson.

Business Dictionary (2016). **Marginal Analysis**. Available: http://www.businessdictionary.com/definition/ marginal-analysis.html. Accessed: 29.05.2016.

Emmett Stuart. (2005). **Exellence in Warehouse Management: How to Minimise Costs and Maximise Value**. 1st edition. Wiley.

Ebeling, Charles E. (1996). **An Introduction to Reliability and Maintainability Engineering**. 1st edition. McGraw-Hill Science/Engineering/Math.

JFukuda (2007). **General Expression Deduction of EBO with Poisson Distribution**. Available: http://www.geocities.ws/riekonissi/ebo.pdf. Accessed: 29.05.2016.

Jones, James V. (2006). **Integrated Logistics Support Handbook**. 3rd edition. McGraw-Hill Education.

Kothari, C. R. (2013). **Research Methodology: Methods and Techniques**. 3rd edition. New Age International Pvt Ltd Publishers.

Python Software Foundation (2016). **Comparing Python to Other Languages.** Available: https://www.python.org/doc/essays/comparisons/. Accessed: 28.05.2016.

Python Wiki (2014). **Beginners Guide: Overview**. Available: https://wiki.python.org/ moin/BeginnersGuide/Overview. Accessed: 28.05.2016.

Severance, Charles R. (2013). **Python for Informatics: Exploring Information**. 1st edition. CreateSpace Independent Publishing Platform.

Sherbrooke, Craig C. (2004). **Optimal Inventory Modeling of Systems: Multi-Echelon Techniques**. 2nd edition. Springer.

Tutorials Point (2016). **Python Variable Types**. Available: http://www.tutorialspoint. com/python/python_variable_types.htm. Accessed: 28.05.2016.

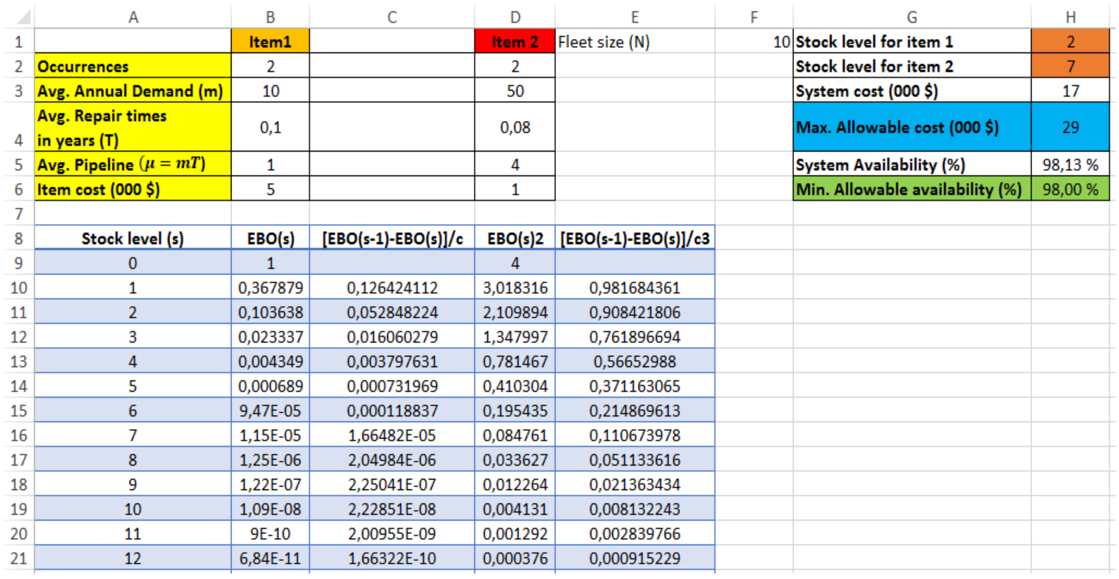
Waters, D. (2003). **Logistics: An Introduction to Supply Chain Management.** Palgrave Macmillan.

Wikipedia (2016). **Consumables**. Available: https://en.wikipedia.org/wiki/ Consumables. Accessed: 29.05.2016.

Wikipedia (2016). **History of Python**. Available: http://en.wikipedia.org/wiki/History _of_Python. Accessed: 28.05.2016.

Appendices

Appendice 1. Model in Excel



	A	B	C	D	E	F	G	H
1		Item1		Item 2	Fleet size (N)	10	Stock level for item 1	2
2	Occurrences	2		2			Stock level for item 2	7
3	Avg. Annual Demand (m)	10		50			System cost (000 $)	17
4	Avg. Repair times in years (T)	0,1		0,08			Max. Allowable cost (000 $)	29
5	Avg. Pipeline ($\mu = mT$)	1		4			System Availability (%)	98,13 %
6	Item cost (000 $)	5		1			Min. Allowable availability (%)	98,00 %
7								
8	Stock level (s)	EBO(s)	[EBO(s-1)-EBO(s)]/c	EBO(s)2	[EBO(s-1)-EBO(s)]/c3			
9	0	1		4				
10	1	0,367879	0,126424112	3,018316	0,981684361			
11	2	0,103638	0,052848224	2,109894	0,908421806			
12	3	0,023337	0,016060279	1,347997	0,761896694			
13	4	0,004349	0,003797631	0,781467	0,56652988			
14	5	0,000689	0,000731969	0,410304	0,371163065			
15	6	9,47E-05	0,000118837	0,195435	0,214869613			
16	7	1,15E-05	1,66482E-05	0,084761	0,110673978			
17	8	1,25E-06	2,04984E-06	0,033627	0,051133616			
18	9	1,22E-07	2,25041E-07	0,012264	0,021363434			
19	10	1,09E-08	2,22851E-08	0,004131	0,008132243			
20	11	9E-10	2,00955E-09	0,001292	0,002839766			
21	12	6,84E-11	1,66322E-10	0,000376	0,000915229			

Formula at B5: = B3 * B4

Formula at D5: = D3 * D4

Formula at B10: = B$5*POISSON.DIST($A10;B$5;0)+(B$5-$A10)*(1-POISSON.DIST($A10;B$5;1))

Formula at C10: = (B9-B10)/$B$6

Formula at D10: = D$5*POISSON.DIST($A10;D$5;0)+(D$5-$A10)*(1-POISSON.DIST($A10;D$5;1))

Formula at E10: = (D9-D10)/$D$6

Formula at H3: = H1*B6+H2*D6

Formula at H5: = ((1-INDEX(Table1;MATCH(H1;Table1[Stock level (s)];0);2)/(F1*B2))^B2)*((1-INDEX(Table1;MATCH(H2;Table1[Stock level (s)];0);4)/(F1*D2))^D2)

Appendice 2. Program Source Code (Written in Python programming language)

```
import math


#Creating base variables
while True:
    N = raw_input('Please enter your fleet size: ')
    try:
        N = int(N)
        N >= 1
        break
    except:
        print 'Wrong input!!! Please type only positive integer (-_-)'
while True:
    numitems = raw_input('Please enter number of critical items in one unit (It should
be greater than 1): ')
    try:
        numitems = int(numitems)
        numitems > 1
        break
    except:
        print 'Wrong input!!! Please type only positive integer (-_-)'
while True:
    length = raw_input('Please enter number of stock level (s) taken into consideration
for each critical item: ')
```

```
    try:
        length = int(length)
        length >= 1
        break
    except:
        print 'Wrong input!!! Please type only positive integer (-_-)'


#Creating a list of EBOs
while True:
    Prac = 0
    t = list()
    for num in range(1,numitems+1):
        while True:
            m = raw_input('Please enter the average annual demand for item ' + str(num)
+ ': ')
            try:
                m = float(m)
                m >= 0
                break
            except:
                print 'Wrong input!!! Please type only positive real numbers (-_-)'
        while True:
            T = raw_input('Please enter the average repair time in years for item ' +
str(num) + ': ')
            try:
```

```
            T = float(T)
            T >= 0
            break
        except:
            print 'Wrong input!!! Please type only positive real numbers (-_-)'
    apl = m*T
    for x in range(length):
        Pr = (apl**x)*math.exp(-apl)/math.factorial(x)
        Prac = Prac + Pr
        ebo = apl*Pr + (apl-x) * (1-Prac)
        t.append(ebo)
    Prac = 0
t = [t[p:p+length] for p in range(0,len(t),length)]
#print t


countitems = 1
count1 = 0
count2 = 0
while True:
    for p in t[count1]:
        print ('Item ' + str(countitems) + ' EBO at s = ' + str(count2) + ' : ' + str(p))
        count2 = count2 + 1
        if count2 > (length-1): break
    if countitems >= numitems or count1 >= (numitems-1): break
```

```
        else:
            countitems = countitems + 1
            count2 = 0
            count1 = count1+1
    print 'Please check the raw data once again !!!'
    print 'Type (Yes) in case you want to continue, (No) to reinput the data.'
    while True:
        ques = raw_input('Continue or not ? ')
        if ques == 'Yes': break
        elif ques == 'No': break
        else:
            print 'Please answer only (Yes) or (No) !!!'
    #print t
    if ques == 'Yes': break

#Creating a list of unit cost for each items
while True:
    costdata = list()
    for p in range(1,numitems+1):
        while True:
            uc = raw_input('Please enter unit cost for item ' + str(p) + ': ')
            try:
                uc = float(uc)
                costdata.append(uc)
            break
        except:
            print 'Wrong input!!! Please enter positive integer (-_-)'
    print 'Please check the unit cost data once again !!!'
    print 'Type (Yes) in case you want to continue, (No) to reinput the data.'
    while True:
        ques = raw_input('Continue or not ? ')
        if ques == 'Yes': break
        elif ques == 'No': break
        else:
            print 'Please answer only (Yes) or (No) !!!'
    #print costdata
    if ques == 'Yes': break


#Creating combination of stock level between items
temp = range(length)*numitems
names = [temp[p:p+length] for p in range(0,len(temp),length)]
#print names
while True:
    if len(names) <= 1: break
    else:
        nametemp = list()
        for n1 in names[0]:
            for n2 in names[1]:
```

```
            name = str(n1) + "&" + str(n2)
            nametemp.append(name)
        names[0:2] = []
        names.insert(0,nametemp)
names = names[0]
#print names


#Creating a cost list for each stock level
i = 0
inp = range(length)*numitems
inp = [inp[p:p+length] for p in range(0,len(temp),length)]
temp = list()
while i < len(inp):
    for c in costdata:
        for t1 in inp[i]:
            cost = c*t1
            temp.append(cost)
        i = i + 1
costlist = [temp[p:p+length] for p in range(0,len(temp),length)]
#print costlist


#Making final cost list for each comabination
while True:
    if len(costlist) <= 1: break
    else:
        temp = list()
        for c1 in costlist[0]:
            for c2 in costlist[1]:
                sum = c1 + c2
                temp.append(sum)
        costlist[0:2] = []
        costlist.insert(0,temp)
costlist = costlist[0]
#print costlist


#Creating a list for occurences
occlist = list()
while True:
    for num in range(1,numitems+1):
        mem = raw_input('Please enter occurences for item ' + str(num) + ': ')
        if mem == 'quit': quit()
        try:
            mem = float(mem)
            occlist.append(mem)
        except:
            print 'Wrong input!!! Please type again (-.-)'
    print 'Please check the occurences data once again !!!'
    print 'Type (Yes) in case you want to continue, (No) to reinput the data.'
    while True:
        ques = raw_input('Continue or not ? ')
        if ques == 'Yes': break
        elif ques == 'No': break
        else:
            print 'Please answer only (Yes) or (No) !!!'
    if ques == 'Yes': break
#print occlist


#Creating a list for system availability
temp1 = list()
ran = range(len(t))
Z = 0
#print i
for p in ran:
    for p1 in t[p]:
        ex = pow(1-(p1/(N*occlist[Z])),occlist[Z])
        temp1.append(ex)
    Z = Z + 1
temp1 = [temp1[p:p+length] for p in range(0,len(temp1),length)]
#print temp1
while True:
    if len(temp1) <= 1: break
    else:
        temp2 = list()
        for p2 in temp1[0]:
            for p3 in temp1[1]:
                ava100 = p2*p3
                temp2.append(ava100)
        temp1[0:2] = []
        temp1.insert(0,temp2)
temp1 = temp1[0]
#print temp1


while True:
    if len(t) <= 1: break
    else:
        temp = list()
        for t1 in t[0]:
            for t2 in t[1]:
                sum = t1 + t2
                temp.append(sum)
        t[0:2] = []
        t.insert(0,temp)
t = t[0]
#print t
```

```
#Creating dictionaries

costdict = dict()

ebodict = dict()

avadict = dict()

optdata = dict()

pos1 = 0

pos2 = 0

pos3 = 0

for d in names:

   ebodict[d] = t[pos1]

   pos1 = pos1 + 1

#print ebodict

for d in names:

   costdict[d] = costlist[pos2]

   pos2 = pos2 + 1

#print costdict

for d in names:

   avadict[d] = temp1[pos3]

   pos3 = pos3 + 1

#print avadict


#Finding optimal inventory policy

while True:

   print 'Type (a) for constraint of availability and (c) for cost'
```

```
quesf = raw_input('Would you like to choose constraint for availability or cost? ')
if quesf == 'c':
  while True:
    ltdcost = raw_input('Please enter maximum allowable cost for spares: ')
    if float(ltdcost) < min(costdict.itervalues()):
      print 'The cost is too small, please enter at least',min(costdict.itervalues())
      continue
    try:
      ltdcost = float(ltdcost)
      optdata1 = dict()
      for key,value in costdict.iteritems():
        if value > ltdcost: continue
        else:
          for key1,value1 in avadict.iteritems():
            if key != key1: continue
            else:
              optdata1[key] = value1
      #print optdata
      com = max(optdata1, key=optdata1.get)
      syscost = costdict[com]
      syscost = round(syscost,3)
      sysebo = ebodict[com]
      sysebo = round(sysebo,3)
      sysava = avadict[com]
```

```
            sysava = round(sysava*100,3)
            com = com.split('&')
            num = 1

            print 'Optimal inventory policy: '
            for p in com:
                print 'Item',num,':',p
                num = num + 1

            print 'System EBO(s):',sysebo
            print ('System Availability: ' + str(sysava) + '%')
            print 'System Cost:',syscost

            while True:
                ques1 = raw_input('Would like to try another allowable cost (Type Yes or
No)? ')
                if ques1 == 'Yes': break
                elif ques1 == 'No': break
                else:
                    print 'Wrong input!!! Please type only Yes or No (-_-)'

        except:
            print 'Wrong input!!! Please type only positive integer (-_-)'
        if ques1 == 'No': break
```

```
  elif quesf == 'a':
    while True:
      ltdava = raw_input('Please enter minimum allowable availability for the fleet:
')
      if float(ltdava) < min(avadict.itervalues()):
        print 'The availability is too small, please enter at least',min(avadict.interval-
ues())
        continue
      try:
        ltdava = float(ltdava)
        optdata2 = dict()
        for key,value in avadict.iteritems():
          if value < ltdava: continue
          else:
            for key1,value1 in costdict.iteritems():
              if key != key1: continue
              else:
                optdata2[key] = value1
        #print optdata
        com = min(optdata2, key=optdata2.get)
        syscost = costdict[com]
        syscost = round(syscost,3)
        sysebo = ebodict[com]
        sysebo = round(sysebo,3)
        sysava = avadict[com]
```

```
            sysava = round(sysava*100,3)

            com = com.split('&')

            num = 1


            print 'Optimal inventory policy: '

            for p in com:

                print 'Item',num,':',p

                num = num + 1


            print 'System EBO(s):',sysebo

            print ('System Availability: ' + str(sysava) + '%')

            print 'System Cost:',syscost


            while True:

                ques1 = raw_input('Would like to try another allowable availability (Type
Yes or No)? ')

                if ques1 == 'Yes': break

                elif ques1 == 'No': break

                else:

                    print 'Wrong input!!! Please type only Yes or No (-_-)'


        except:

            print 'Wrong input!!! Please type only positive integer (-_-)'

        if ques1 == 'No': break
```

```
else:
    print 'Wrong input!!! Please type only (a) or (c) (-_-))'
    continue
while True:
    quesc = raw_input('Would you like to change the constraint? ')
    if quesc == 'Yes':
        break
    elif quesc == 'No': break
    else:
        print 'Wrong input!!! Please type only Yes or No (-_-)'
if quesc == 'No': break
```