DAILY ASSESSMENT FORMAT

|  |  |  |  |
| --- | --- | --- | --- |
| Date: | 19/05/2020 | Name: | Shilpa N |
| Course: | TCS ion | USN: | 4AL16EC071 |
| Topic: | |  | | --- | | Gain Guidance from Career Gurus | | Write a Winning Resume and Cover Letter | | Stay Ahead in Group Discussions | | Semester & Section: | 8 “B” |
| Github Repository: | Shilpan-test |  |  |

|  |
| --- |
| FORENOON SESSION DETAILS |
| Image of session |
| Report – Report can be typed or hand written for up to two pages.  **PRESENTATION**  Today the module gave the brief information about how to be more effective while giving a presentation. The objective of the module is to design effective ppt with DO’s and don’t’s in the presentation skills. The effective presentation should have known what, who, why, where and when. The module also explained about the design of the slides the more complex it gets and the attention of the participants is lost, keep the number of slides to the minimum. Further the module explained about what should be avoided while presentation that the language used should be simple, and understood by all, abbreviation words should be used only after using the full form for the first time and the module also explained about the alignment of images in the slide. For powerful presentation quotations should be added this helps to reduce the number of slides ang gets the audience thinking, and never forget to give credit to the author. In the final slide the presentation should contain FAQ relevant to the topic along with end and thank you slide. Further, the font size of the slide should be readable and never express the sentence in upper cases. The slides should be attractive based on the audience  **SOFT SKILL DEVELOPMENT**  The module explained about the necessary soft skill one should have while working in a group. The chapter explains the importance of soft skills and distinguish between the hard and soft skills. Basically, soft skills are people skills that arr difficult to quantify and measure. These skills help to maintain healthy relationship. the soft skill includes communication skills, time management, negotiation skills, critical thinking, self-confidence, business etiquette, goal setting and team work. In a group we should possess following qualities that is express thoughts, work in a team, interview handling, ability to empathize, self-awareness, learning and sharing, good work ethics and interpersonal skills. The module differentiated between hard and soft skills.  **WRITING WINNING RESUME AND COVER LETTER:**  The objective of the session is to create own resume and do’s & don’ts in writing a resume. How to draft a cover letter. The importance of resume & its structure is explained. How resume is important in selecting for a job as it is first impression. The resume also has a power to get a dream job and it can make us not selected for a job. Most of the companies initially select candidates based on the resume. How to create structure of resume is explained. Types of resume-chronological, functional and combinational resume. Resume should have bullet points. Use formal readable fonts. Should be cautious while writing hobbies. Don’t exceed more than 2pages. Don’t lie about achievements. Don’t copy things from internet. Cover letter is a must should be attached to the resume whether it may be email. Cover letter is read before resume, it tells employer about your skill to the particular position. Structure of cover letter is discussed. Identify skill relevant to job status. check for the grammatic mistakes. Resume should crisp to the point. Do not lie in resume. Resume is a you on a sheet of paper.  **STAY AHEAD IN GROUP DISCUSSION:**  Here we are going to learn how effectively we have to present our self in group discussion. GD is part of interview process has specific intention. The GD is often use as mass grading process. All the members are expected to share their views. GD is conducted to check interpersonal skill, overall skill. Posture, gesture, eye contact has greater impact than words. Make sure your view is clear. Listening to others help in getting point on the topic. Maintain a neutral tone. Don’t use slangs. Don’t monk at others. Get to the point while expressing your thought. don’t lose your confidence, listen well and boost yourself. Dress formally. Maintain eye contact with others. Listen to all the points made. Try to summarize. Keep a track on time. Don’t interrupt. Avoid emotional outbreak. Don’t be biased. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Date: | 19/05/2020 | Name: | Shilpa N | |
| Course: | PYTHON | USN: | 4AL16EC071 | |
| Topic: | |  | | --- | | 1. List Comprehensions | | 1. More on Functions | | 1. File Processing | | 1. Imported Modules | | Semester & Section: | 8 “B” | |
| AFTERNOON SESSION DETAILS | | | |
| **List Comprehensions:**  Python is popular for permitting you to compose code that is exquisite, simple to compose, and nearly as simple to peruse as plain English. One of the language's most particular highlights is the list comprehensions, which you can use to make amazing usefulness inside a solitary line of code. Be that as it may, numerous engineers battle to completely use the further developed highlights of a list comprehensions in Python. A few software engineers even use them to an extreme, which can prompt code that is less productive and harder to peruse.  list comprehensions are a third method of making lists. With this elegant approach, you could rewrite the for loop from the first example in just a single line of code:  >>>  >>> squares = [i \* i for i in range(**10**)]  >>> squares  [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]  Rather than creating an empty list and adding each element to the end, you simply define the list and its contents at the same time by following this format:  >>>  new\_list = [expression for member in iterable]  Every list comprehension in Python includes three elements:   1. **expression** is the member itself, a call to a method, or any other valid expression that returns a value. In the example above, the expression i \* i is the square of the member value. 2. **member** is the object or value in the list or iterable. In the example above, the member value is i. 3. **iterable** is a list, [set](https://realpython.com/python-sets/), sequence, [generator](https://realpython.com/introduction-to-python-generators/), or any other object that can return its elements one at a time. In the example above, the iterable is range(10).   Because the expression requirement is so flexible, a list comprehension in Python works well in many places where you would use map(). You can rewrite the pricing example with its own list comprehension:  >>>  >>> txns = [1.09, 23.56, 57.84, 4.56, 6.78]  >>> TAX\_RATE = .**08**  >>> def get\_price\_with\_tax(txn):  ... return txn \* (**1** + TAX\_RATE)  >>> final\_prices = [get\_price\_with\_tax(i) for i in txns]  >>> final\_prices  [1.1772000000000002, 25.4448, 62.467200000000005, 4.9248, 7.322400000000001]  The only distinction between this implementation and map() is that the list comprehension in Python returns a list, not a map object. Benefits of Using List Comprehensions List comprehensions are regularly portrayed as being more Pythonic than loops or map(). But instead than indiscriminately tolerating that evaluation, it's justified, despite all the trouble to comprehend the advantages of utilizing a list comprehension in Python when contrasted with the other options. Later on, you'll find out around a couple of situations where the choices are a superior decision.One main benefit of using a list comprehension in Python is that it’s a single tool that you can use in many different situations. In addition to standard [list creation](https://realpython.com/python-lists-tuples/), list comprehensions can also be used for mapping and filtering. You don’t have to use a different approach for each scenario.  This is the main reason why list comprehensions are considered Pythonic, as Python embraces simple, powerful tools that you can use in a wide variety of situations. As an added side benefit, whenever you use a list comprehension in Python, you won’t need to remember the proper order of arguments like you would when you call map().  List comprehensions are also more declarative than loops, which means they're simpler to peruse and comprehend. loops expect you to concentrate on how the list is made. You need to physically make an empty list, loop over the components, and add each of them to the end of the list. With a list comprehension in Python, you can rather concentrate on what you need to go in the list and trust that Python will deal with how the list construction happens.  **File processing**  file is a contiguous set of bytes used to store data. This data is organized in a specific format and can be anything as simple as a text file or as complicated as a program executable. In the end, these byte files are then translated into binary 1 and 0 for easier processing by the computer.  Files on most modern file systems are composed of three main parts:   1. Header: metadata about the contents of the file (file name, size, type, and so on) 2. Data: contents of the file as written by the creator or editor 3. End of file (EOF): special character that indicates the end of the file  Opening and Closing a File in Python When you want to work with a file, the first thing to do is to open it. This is done by invoking the open() built-in function. open() has a single required argument that is the path to the file. open() has a single return, the file object:  file = open('dog\_breeds.txt') Text File Types A text file is the most common file that you’ll encounter. Here are some examples of how these files are opened:  open('abc.txt')  open('abc.txt', 'r')  open('abc.txt', 'w')  With these types of files, open() will return a TextIOWrapper file object:  >>>  >>> file = open('dog\_breeds.txt')  >>> type(file)  <class '\_io.TextIOWrapper'>  This is the default file object returned by open(). Buffered Binary File Types A buffered binary file type is used for reading and writing binary files. Here are some examples of how these files are opened:  open('abc.txt', 'rb')  open('abc.txt', 'wb')  With these types of files, open() will return either a BufferedReader or BufferedWriter file object:  >>>  >>> file = open('dog\_breeds.txt', 'rb')  >>> type(file)  <class '\_io.BufferedReader'>  >>> file = open('dog\_breeds.txt', 'wb')  >>> type(file)  <class '\_io.BufferedWriter'> Raw File Types A raw file type is:  “generally used as a low-level building-block for binary and text streams.” ([Source](https://docs.python.org/3.7/library/io.html#raw-i-o))  It is therefore not typically used.  Here’s an example of how these files are opened:  open('abc.txt', 'rb', buffering=0)  With these types of files, open() will return a FileIO file object:  >>>  >>> file = open('dog\_breeds.txt', 'rb', buffering=0)  >>> type(file)  <class '\_io.FileIO'>  A module allows you to logically organize your Python code. Grouping related code into a module makes the code easier to understand and use. A module is a Python object with arbitrarily named attributes that you can bind and reference.  Simply, a module is a file consisting of Python code. A module can define functions, classes and variables. A module can also include runnable code. Example The Python code for a module named aname normally resides in a file named aname.py. Here's an example of a simple module, support.py  def print\_func( par ):  print "Hello : ", par  return The import Statement You can use any Python source file as a module by executing an import statement in some other Python source file. The import has the following syntax −  import module1[, module2[,... moduleN]  When the interpreter encounters an import statement, it imports the module if the module is present in the search path. A search path is a list of directories that the interpreter searches before importing a module. For example, to import the module support.py, you need to put the following command at the top of the script −  #!/usr/bin/python  # Import module support  import support  # Now you can call defined function that module as follows  support.print\_func("Zara")  When the above code is executed, it produces the following result −  Hello : Zara  A module is loaded only once, regardless of the number of times it is imported. This prevents the module execution from happening over and over again if multiple imports occur. The from...import Statement Python's from statement lets you import specific attributes from a module into the current namespace. The from...import has the following syntax −  from modname import name1[, name2[, ... nameN]]  For example, to import the function fibonacci from the module fib, use the following statement −  from fib import fibonacci  This statement does not import the entire module fib into the current namespace; it just introduces the item fibonacci from the module fib into the global symbol table of the importing module. The from...import \* Statement It is also possible to import all names from a module into the current namespace by using the following import statement −  from modname import \*  This provides an easy way to import all the items from a module into the current namespace; however, this statement should be used sparingly. Locating Modules When you import a module, the Python interpreter searches for the module in the following sequences −   * The current directory. * If the module isn't found, Python then searches each directory in the shell variable PYTHONPATH. * If all else fails, Python checks the default path. On UNIX, this default path is normally /usr/local/lib/python/.   The module search path is stored in the system module sys as the **sys.path** variable. The sys.path variable contains the current directory, PYTHONPATH, and the installation-dependent default. The PYTHONPATH Variable The PYTHONPATH is an environment variable, consisting of a list of directories. The syntax of PYTHONPATH is the same as that of the shell variable PATH.  Here is a typical PYTHONPATH from a Windows system −  set PYTHONPATH = c:\python20\lib;  And here is a typical PYTHONPATH from a UNIX system −  set PYTHONPATH = /usr/local/lib/python Namespaces and Scoping Variables are names (identifiers) that map to objects. A namespace is a dictionary of variable names (keys) and their corresponding objects (values).  A Python statement can access variables in a local namespace and in the global namespace. If a local and a global variable have the same name, the local variable shadows the global variable.  Each function has its own local namespace. Class methods follow the same scoping rule as ordinary functions.  Python makes educated guesses on whether variables are local or global. It assumes that any variable assigned a value in a function is local.  Therefore, in order to assign a value to a global variable within a function, you must first use the global statement.  The statement global VarName tells Python that VarName is a global variable. Python stops searching the local namespace for the variable.  For example, we define a variable Money in the global namespace. Within the function Money, we assign Money a value, therefore Python assumes Money as a local variable. However, we accessed the value of the local variable Money before setting it, so an UnboundLocalError is the result. Uncommenting the global statement fixes the problem.  #!/usr/bin/python  Money = 2000  def AddMoney():  # Uncomment the following line to fix the code:  # global Money  Money = Money + 1  print Money  AddMoney()  print Money The dir( ) Function The dir() built-in function returns a sorted list of strings containing the names defined by a module.  The list contains the names of all the modules, variables and functions that are defined in a module. Following is a simple example −  #!/usr/bin/python  # Import built-in module math  import math  content = dir(math)  print content  When the above code is executed, it produces the following result −  ['\_\_doc\_\_', '\_\_file\_\_', '\_\_name\_\_', 'acos', 'asin', 'atan',  'atan2', 'ceil', 'cos', 'cosh', 'degrees', 'e', 'exp',  'fabs', 'floor', 'fmod', 'frexp', 'hypot', 'ldexp', 'log',  'log10', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh',  'sqrt', 'tan', 'tanh']  Here, the special string variable \_\_name\_\_ is the module's name, and \_\_file\_\_ is the filename from which the module was loaded. | | | |