Resource and Scheduling Tool User Guide

# Overview

A resource and scheduling tool that aims to streamline a few work processes (which requires connection between multiple excel sheets) in a manufacturing environment.

The primary aim is to make sure all jobs have adequate resources allocated to it and the job can be completed before a stipulated deadline or a date recommendation (for the deadline) will be provided based on current resource available.

Submission Content:

Submission contains 5 files in total, as described below:

**Two .py scripts:**

1. main\_code\_base.py
2. class\_functions.py (script containing all the user defined functions and classes)

**Two .csv files\*:**

1. employee.csv (an eg. Of Employee details a company might have)
2. job.csv (an example of scheduled jobs that a company might have)

*\*Note: The .csv files should not be edited as the intent of these files is to “act” as an initial database to load data and assumes that the employee and jobs data are accurate and in the correct format.*

**One .docx file:**

1. Scheduling Program User Guide

# Set up

Description of context:

All files above (described in Submission content) must be located within the same file path, due to connections between the .py scripts and loading of data from .csv files.

The .csv files act as initial data for initialization of the Tool, with the following data columns described below:

The employee.csv contains information of each employee using the following columns:

* Employee ID – integers only
* First Name
* Last Name
* Hourly Rate – numerical values (SGD$)
* Total Hours per day – numerical values, describing the number of hours an employee has
* Competency – numerical values
* Craft – “Machinery”, “Metals”, or “Instrument/Electrical”

The job.csv database contains information of each job using the following columns:

* Job Name
* Start Date – must be in the format of dd/mm/yyyy, e.g. 15/2/2023
* Completion by Date – must be in the format of dd/mm/yyyy
* Resources required – in hours, numerical values only
* Total Cost – numerical values (SGD$)
* Craft - can only be “Machinery”, “Metals”, or “Instrument/Electrical”

These jobs are accurate in terms of the checks between Resources required and Total Hours Per Day in the employee.csv by craft and hence can be scheduled when the Tool is initialised

Set-up steps:

1. Place all five submission files in the same folder
2. On the Terminal or command line, run script “main\_code\_base”.py
3. Do these initialisation steps (in yellow) 🡪 Option 1: Upload Employee/Job Database [From .CSV only] 🡪 1: Employee database 🡪 Input “Y” to the prompt: Do you want to proceed with another action? Y/N
4. The Calendar Resource has been initialised, **please proceed with Test Cases below**
5. (Optional) Once step 1- 4 is completed, initialising of job database is also available through these steps: 🡪 Option 1: Upload Employee/Job Database [From .CSV only] 🡪 2: Job database 🡪 Input “Y” to the prompt: Do you want to proceed with another action? Y/N

Once the above steps are done, resource and scheduling tool is ready for use and the tool works for the date range of 2022-12-31 to 2042-12-31.

Note: This date range can be extended further based on the datetime in-built python module, however, this is not our intention at this point.

# Classes

\*Classes are defined in the class\_functions.py file, there are a total of two classes

1.Job Class (Public Class)

Attributes:

* job\_id
* job\_name
* start\_date
* due\_date (original planned due date/deadline inputted by User **or** the recommended due date based on resource available after function scheduleJobCheck is called)
* resources
* total\_cost
* employees (this is an association relationship with the employee class)
* craft
* scheduled\_end\_date (This is the actual end date of the job scheduled in the system, not necessarily the same as the user input due date)

Methods:

Nil

2.Employee Class (Private Class)

Attributes:

* emp\_id (Employee ID)
* first\_name
* last\_name
* hourly\_rate
* total\_hours\_per\_day
* competency
* craft

Methods:

* getter/setter methods for all attributes above (instance method)
* CurrentEmployeeList (static method)
* addEmployee (static method)
* removeEmployee (static method)

# Functions

Overview of Functions defined in class\_functions.py:

1. createCalendarRange (start\_date, end\_date, calendar\_resource\_dict, list\_of\_employees)
   1. Takes in 4 parameters and creates a calendar data structure if calendar\_resource\_dict is an empty dictionary
   2. Calendar data structure is of three-dimesions of this format:

{start\_date: [{emp\_id: total\_work\_hours\_per\_day, “Craft” : Employee’s Craft}], start\_date +1 (day) : [{emp\_id: total\_work\_hours\_per\_day, “Craft” : Employee’s Craft }],end\_date: [{emp\_id: total\_work\_hours\_per\_day, “Craft” : Employee’s Craft } ] }

* 1. Calendar data structure runs from start\_date all the way till end\_date
  2. Returns calendar data structure

1. scheduleJobCheck (job\_name, start\_date, due\_date, resources, total\_cost, craft, calendar\_resource\_dict)
   1. Takes in 7 parameters which describes the job requirements (start and end date and the craft and resources required)
   2. These requirements are checked against calendar\_resource\_dict to see if there are sufficient available resources to complete the job before or on the end date
   3. If sufficient available resource, prompt user to confirm schedule, returns True and the job dates if user confirms, else return False
   4. Else, prompt user if user would want a best available schedule recommended by the tool
   5. If user wants a best available schedule, call recommendSchedule function and prompts user if user wants to accept the new dates returned from recommendSchedule. Returns True and new dates if user agrees, else return False.
   6. Else, return False
2. recommendSchedule(resources, start\_date, due\_date, craft, calendar\_resource\_dict)
   1. Takes in 5 parameters containing job requirements of start\_date, due\_date and resources and craft required
   2. Compares these requirements with calendar\_resource\_dict and saves the start and end date where this job can be fulfilled completely
   3. Returns start date and new end date if the variables are different, else return new end date and None
3. scheduleJob(job\_name, start\_date, due\_date, resources, total\_cost, craft ,calendar\_resource\_dict, current\_job\_id, list\_of\_jobs)
   1. Takes in 9 parameters
   2. Generates a Job ID for current job being scheduled
   3. Creates new job instance and saves Job instance in list\_of\_jobs
   4. Allocate Employees in calendar\_resource\_dict to the job
   5. Creates an association between new job instance with the employees allocated to it
   6. Sets Job intance scheduled\_end\_date attribute to actual end\_date in the system (might be different from the due\_date originally given by user as some jobs can be completed earlier)
   7. Output Job details scheduled in the system for User, details shared are the start date and Job ID, as well as employees tagged to the job on the various dates

# Use Cases

## Upload employee/job database

### Upload employee database

This must be the first step to do before the scheduling program can work.

If “employee.csv” is present in the same folder with the right format, a list of employee object will be created and stored in the variable list\_of\_employees.

At the same time, a calendar in the form of dictionary – calendar\_resource\_dict will be created from 31 Dec 2022 to 31 Dec 2042, the calendar contains the available resource each day in the duration stated above in the below data structure.

### Upload job database

This function allows previously scheduled job tabulated in the .csv format to be imported by the program. The calendar created in 5.1.1 will be updated to reflect the resources availability change in the respective dates. It should be the second step that needs to be completed and the rest of function can work based on the information provided.

If “job.csv” is present in the same folder with the right format, a list of employee object will be created and stored in the variable list\_of\_jobs.

## Add/Remove Employees or Update Job(s)

### Add an employee to database

User needs to input the following details, separated by commas: Employee ID, First Name, Last Name, Hourly Rate, Total Hours Per Day, Competency, Craft, Employee Start Date in yyyy-mm-dd.

Once the inputs are validated, the calendar resource dictionary will be updated to add the new employee after his/her start date.

|  |  |
| --- | --- |
| Sample input | Sample output |
|  |  |
|  |  |
|  |  |
|  |  |

### Remove an employee from database

User needs to input Employee ID and Last Day of Work in yyyy-mm-dd with commas separating each input.

Once the inputs are validated, the employee will be removed from the calendar resource dictionary after his/her last day of work.

|  |  |
| --- | --- |
| Sample input | Sample output |
|  |  |
|  |  |
|  |  |
|  |  |

### Delete a scheduled job

User needs to input the job ID number. The program will remove the job from the list\_of\_jobs list and restore the calendar resource dictionary.

|  |  |
| --- | --- |
| Sample input | Sample output |
|  |  |
|  |  |

## Schedule a job

User needs to input the following details, separated by commas: Job Name, Start Date in yyyy-mm-dd, Due Date in yyyy-mm-dd, Resources Required in hours, Total cost, Craft Required

The program will call for the *scheduleJobCheck* function and check the earliest possible day where workers from the right craft has hours available to fulfil the resources required, and whether job can be completed before its due date given the earliest possible start date. If it is feasible, program will prompt user to confirm before proceeding for actual job scheduling.

The actual job scheduling is conducted via the *scheduleJob* function, where the required resource is subtracted from the available hours of the respective craft workers in the calendar dictionary. The program will output the job number, job name and the resource requirement.

|  |  |
| --- | --- |
| Sample input | Sample output |
|  |  |
|  |  |
|  |  |
|  |  |

## Calculate Key Performance Indicators

### Find all job details based on a specific date or by date range

User needs to input the either a single date in the format of yyyy-mm-dd, or a date range between 2 dates separated by commas in the same format.

|  |  |
| --- | --- |
| Sample input | Sample output |
|  |  |
|  |  |

### Find job details based on job ID

User needs to input the job ID number. The program will print the job ID, job name, resource required, total project cost and the craft involved. The detailed schedule and the employees involvement on each day is printed as well.

|  |  |
| --- | --- |
| Sample input | Sample output |
|  |  |
|  |  |

### Total Cost spent on Jobs for a specified Date Range

### Total Employee count by Craft for a specified Date

## Additional functions

* Print out the work lineup and free hours for a particular employee in the next 1 month
* Putting rubrics here for reference. To remove before submission

|  |
| --- |
| Problem can only be feasibly solved with computer programming with enhanced productivity and effectiveness  Sufficient use cases to cover affected processes and additional use cases for advanced scenarios  Use cases adopt proper algorithmic thinking  Usage of appropriate basic Python data structures and user-defined data structures  More advanced searching and soring algorithms or attempt to optimize algorithms  Data persistence to files  Appropriate CUI with prompts, cues, feedback message and input data validation |