**Test identification:** Create Event (create_task function)
**Description:** This test suite details the test for creating an event for the calendar
**Tester:** Nigel Tan (32881509)
**Technique used:** Modified Condition/Decision Coverage (MC/DC)
**Cases identified:**

**Input Variable**

- V1: ID must be at least 6 characters long (T for True and F for False)
- V2: Attendee's email must be valid (T for True and F for False)
- V3: End date must not be before the Start date (T for True and F for False)

**Output Variable**

- V4: Error: Invalid Email (Fail)
- V5: Pass: Create Event window closed and the event is added to google calendar (True)
- V6: Error: End date more than start date (Fail)
- V7: Error: Invalid ID (Fail)

| Test | V1 | V2 | V3 | Outcome |
|---|---|---|---|---|
| 1 | F | F | F | V7 |
| 2 | F | F | T | V7 |
| 3 | F | T | F | V7 |
| 4 | F | T | T | V7 |
| 5 | T | F | F | V4 |
| 6 | T | F | T | V6 |
| 7 | T | T | F | V6 |
| 8 | T | T | T | V5 |

V1
Test: {1,5}, {2,6}, {3,7}, **{4,8}**
V2
Test: **{5,7}**
V3
Test: None
Full MC/DC Coverage: {4,5,7,8}

**Justification**

Checks all possible conditions which can go to different lines of codes. By testing all possible boolean conditions, I will have tested the entire function then I reduced the amount of test cases needed to achieve full coverage.

**Test identification:** Task details (task_details function)
**Description:** This test suite details the test for task details for the calendar
**Tester:** Nigel Tan (32881509)
**Technique used:** Modified Condition/Decision Coverage (MC/DC)
**Cases identified:**

**Input Variable**

V1: Calendar event with attendees
V2: User of application is  the attendee
V3: Attendee did not accept or reject yet

**Output Variable**

V4: There is an accept and reject button as well as a delete button (True)
V5: Only delete button (False)

| Test | V1 | V2 | V3 | Outcome |
|---|---|---|---|---|
| 1 | F | F | F | V5 |
| 2 | F | F | T | V5 |
| 3 | F | T | F | V5 |
| 4 | F | T | T | V5 |
| 5 | T | F | F | V5 |
| 6 | T | F | T | V4 |
| 7 | T | T | F | V5 |
| 8 | T | T | T | V5 |

V1
Test: {1,5},{2,6},{3,7},{4,8} = {2,6}
V2
Test: {1,3},{2,4},{5,7},{6,8} = {6,8}
V3
Test: {1,2},{3,4},{5,6},{7,8} = **{5,6}**

Full MC/CD coverage: {5,6}

**Justification**

Checks all possible conditions which can go to different lines of codes. By testing all possible boolean conditions, I will have tested the entire function then I reduced the amount of test cases needed to achieve full coverage.

**Test identification:** Delete task (delete_task function)
**Description:** This test suite details the test for task deletion for the calendar
**Tester:** Nigel Tan (32881509)
**Technique used:** Modified Condition/Decision Coverage (MC/DC)
**Cases identified:**

**Input Variables**
V1: date_start >= date_now
V2: date_end >= date_now
**Output Variables**
V3: Event is cancelled and added to archive (True)
V4: Event is deleted (False)

| Test | V1 | V2 | Outcome |
|---|---|---|---|
| 1 | F | F | V1 |
| 2 | F | T | V3 |
| 3 | T | F | V3 |
| 4 | T | T | V3 |

V1
Test: {1,3} = **{1,3}**

V2
Test: {2,4} = None

Full MC/DC Coverage = {1,3}

**Justification**
Checks all possible conditions which can go to different lines of codes. By testing all possible boolean conditions, I will have tested the entire function then I reduced the amount of test cases needed to achieve full coverage.

**Test identification:** Restore task (restore_event function)
**Description:** This test suite details the test for task restoration for the calendar
**Tester:** Nigel Tan (32881509)
**Technique used:** Modified Condition/Decision Coverage (MC/DC)
**Cases identified:**

**Input Variables**
V1: The event is still in google calendar

**Output Variables**
V2: Event successfully restored (True)
V3: Error: Restore event failed because the event is in google calendar (False)

| Test | V1 | Outcome |
|---|---|---|
| 1 | F | V3 |
| 2 | T | V2 |

V1
Test: **{1,2}**

Full MC/CD Coverage = {1,2}

**Justification**
Checks all possible conditions which can go to different lines of codes. By testing all possible boolean conditions, I will have tested the entire function then I reduced the amount of test cases needed to achieve full coverage.

**Test identification:** Import JSON (import_json function)
**Description:** This test suite details the test for input JSON for the calendar
**Tester:** Nigel Tan (32881509)
**Technique used:** Modified Condition/Decision Coverage (MC/DC)
**Cases identified:**

**Input Variables**
V1: file_ext == "json"
V2: Valid format inside the  JSON file for the program to read properly

**Output Variables**
V3: Error: Input file type invalid (False)
V4: Error: JSON content invalid (False)
V5: Successfully input JSON content (True)

| Test | V1 | V2 | Outcome |
|---|---|---|---|
| 1 | F | F | V3 |
| 2 | F | T | V3 |
| 3 | T | F | V4 |
| 4 | T | T | V5 |

V1
Test: **{1,3}**

V2
Test: **{2,4}**

Full MC/DC coverage: {1,2,3,4}

**Justification**
Checks all possible conditions which can go to different lines of codes. By testing all possible boolean conditions, I will have tested the entire function then I reduced the amount of test cases needed to achieve full coverage.

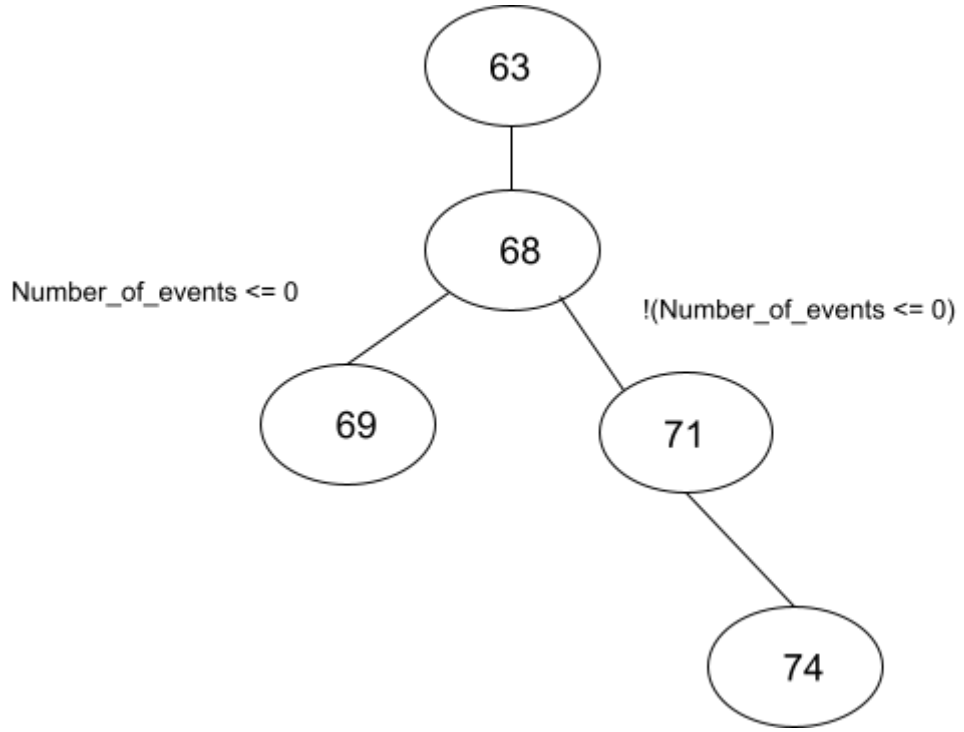
**Test identification**: get_upcoming_events
**Tester** - Yew Yee Perng (32205481)
**Technique used** - path coverage

Conditions - **A** => number_of_events <= 0

| Test | A | outcome |
|---|---|---|
| 1 | T | T |
| 2 | F | F |

**Control flow diagram (CFG)**

63 in the oval means line 63 in the MyEventManager.py code




Justification

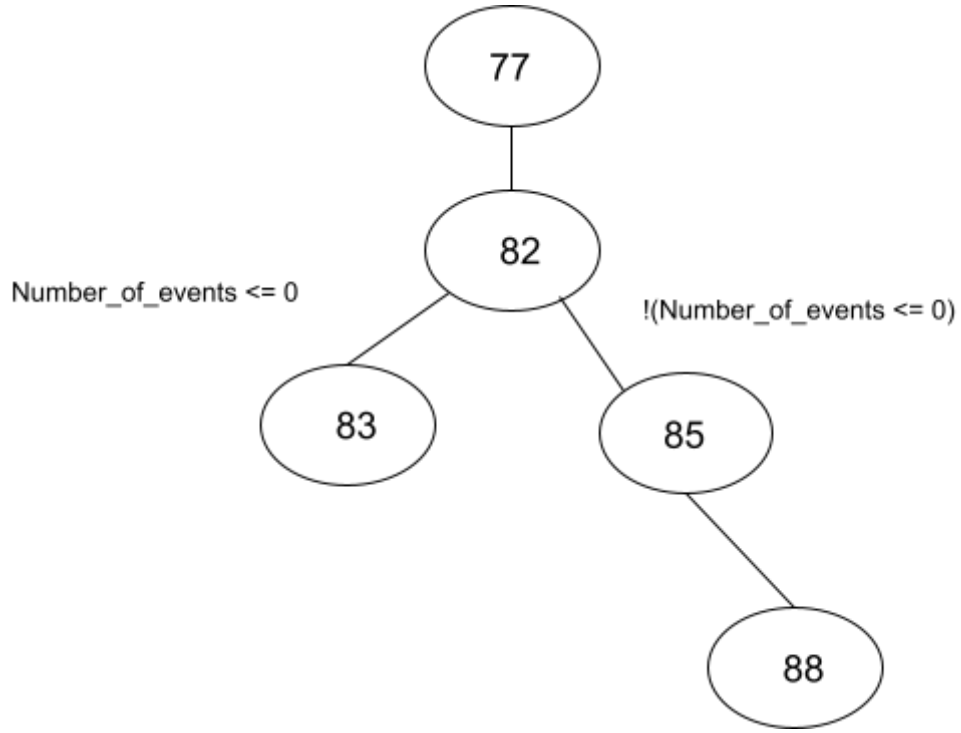
Path coverage is used on this particular test case. There are multiple paths/ branches that could be led to due to different conditions. For this case, there is a if statement in the code which could lead to different outcomes. Therefore, path coverage is suitable to be used as it could clearly show the paths and different outcomes of different paths taken.

**Test Identification**: get_events
**Tester** - Yew Yee Perng (32205481)
**Technique used** - path coverage

Conditions - A => number_of_events <= 0

| Test | A | outcome |
|---|---|---|
| 1 | T | T |
| 2 | F | F |

**Control flow diagram (CFG)**
77 in the oval means line 77 in the MyEventManager.py code



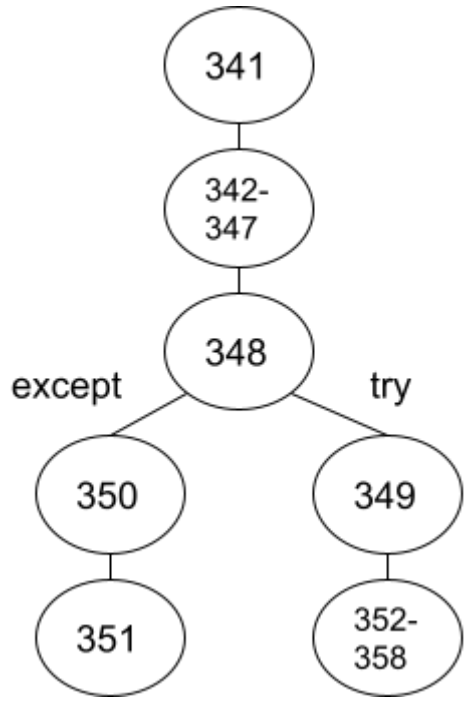

Justification
Path coverage is used on this particular test case. There are multiple paths/ branches that could be led to due to different conditions. For this case, there is a if statement in the code which could lead to different outcomes. Therefore, path coverage is suitable to be used as it could clearly show the paths and different outcomes of different paths taken.

**Test identification:** print_details
**Tester** - Yew Yee Perng (32205481)
**Technique used** - path coverage

condition - A => e6 = Label(searchWind, text = "Event Attendees: " + str(event['attendees']))

| Test | A | outcome |
|---|---|---|
| 1 | T | T |
| 2 | F | F |

**Control flow diagram (CFG)**
341 in the oval means line 341 in the MyEventManager.py code




Justification
Path coverage is used on this particular test case. There are multiple paths/ branches that could be led to due to different conditions. For this case, there is a try and except statement in the code which could lead to different outcomes. Therefore, path coverage is suitable to be used as it could clearly show the paths and different outcomes of different paths taken.

**Test case** - updating_tasks
**Tester** - Yew Yee Perng (32205481)
**Technique used** - Statement coverage
**Parameters used for checking** - updating_tasks(archive, tk, temp, clicked , api)

```
def updating_tasks(archive, tk, temp, clicked, api):
    year = temp.get()
    month = clicked.get()
    last_day = calendar.monthrange(int(year), int(month))[1]
    start_date = date(int(year), int(month), 1)
    end_date = date(int(year), int(month), last_day)
    start_time = str(start_date) + "T00:00:00Z"
    end_time = str(end_date) + "T00:00:00Z"
    events = get_events(api, start_time, end_time, 100)
    populate(events, archive, tk, api)
    return 0
```

**Justification**
By looking through the code, there are no loops in the code. Therefore, with the parameters updating_tasks(archive, tk, temp, clicked , api) where archive is the storage for deleted events, tk is the tkinter constructor, temp is just a holder for a specific value, clicked is a list of dropdown events and api is just our google_calendar_api. Thus, updating_tasks(archive, tk, temp, clicked , api) gives us 100 % of code coverage.

**Test case** - open_form
**Tester** - Yew Yee Perng (32205481)
**Technique used** - Statement coverage
**Parameters used for checking** - open_form(tk , api)

```
def openForm(tk, api):
    newWind = Toplevel(tk)
    newWind.title("Create Event")
    l1 = Label(newWind ,text = "Event ID").grid(row = 0,column = 0)
    l2 = Label(newWind ,text = "Event Name").grid(row = 1,column = 0)
    l3 = Label(newWind ,text = "Event Location").grid(row = 2,column = 0)
    l4 = Label(newWind ,text = "Event Attendees email").grid(row = 3,column = 0)
    l5 = Label(newWind ,text = "Start Date (yyyy-mm-dd)").grid(row = 4,column = 0)
    l6 = Label(newWind ,text = "End Date (yyyy-mm-dd)").grid(row = 5,column = 0)

    maxdate1 = date(2050,12,31)

    id = Entry(newWind)
    id.grid(row = 0,column = 1)
    name = Entry(newWind)
    name.grid(row = 1,column = 1)
    loc = Entry(newWind)
    loc.grid(row = 2,column = 1)
    att = Entry(newWind)
    att.grid(row = 3,column = 1)
    start = DateEntry(newWind, date_pattern='yyyy-mm-dd', maxdate = maxdate1)
    start.grid(row = 4,column = 1)
    end = DateEntry(newWind, date_pattern='yyyy-mm-dd', maxdate = maxdate1)
    end.grid(row = 5,column = 1)
    submitbtn = ttk.Button(newWind, text="Submit", command= lambda: create_task(newWind, api, id, name, loc,att,start, end)).grid(row = 6,column = 1)
    return newWind.state()
```

**Justification**

From looking through the code, we can see that there are no loops. Therefore, with the parameters open_form(tk, api) where tk is the tkinter constructor and api is just the google_calender_api. Therefore, open_form(tk, api) gives us 100% code coverage.

**Test case** - accept_invite
**Tester** - Yew Yee Perng (32205481)
**Technique used** - Statement coverage
**Parameters used for checking** - accept_invite(detailwind ,id, i,  api)

```
def accept_invite(detailWind, id, i, api):
    event = api.events().get(calendarId='primary', eventId=id).execute()
    event['attendees'][i]['responseStatus'] = 'accepted'
    api.events().update(calendarId='primary', eventId=id, body=event).execute()
    detailWind.destroy()
    return 0
```

**Justification**
From looking through the code, there's no loops in this piece of code. Therefore, with the parameters accept_invite(detailwind ,id, i,  api) where detailwind is a top level of tkinter window and id is basically the event id , i is the index of the event and lastly api is just the google_calendar_api. Thus, accept_invite(detailwind ,id, i,  api) gives us 100% of the code coverage.

**Test case** - reject_invite
**Tester** - Yew Yee Perng (32205481)
**Technique used** - Statement coverage
**Parameters used for checking** - reject_invite(detailwind ,id, i,  api)

```
def reject_invite(detailWind, id, i, api):
    event = api.events().get(calendarId='primary', eventId=id).execute()
    event['attendees'][i]['responseStatus'] = 'declined'
    api.events().update(calendarId='primary', eventId=id, body=event).execute()
    detailWind.destroy()
    return 0
```

**Justification**
From looking through the code, there's no loops in this piece of code. Therefore, with the parameters reject_invite(detailwind ,id, i,  api) where detailwind is a top level of tkinter window and id is basically the event id , i is the index of the event and lastly api is just the google_calender_api. Thus, reject_invite(detailwind ,id, i,  api) gives us 100% of the code coverage.

**Test case** - search_form
**Tester** - Yew Yee Perng (32205481)
**Technique used** - Statement coverage
**Parameters used for checking** - search_form(tk , api)

```
def search_form(tk, api):
    searchWind = Toplevel(tk)
    searchWind.title("Search Window")
    searchWind.geometry("800x500")

    l1 = Label(searchWind ,text = "Search by Event Name: ")
    search_term = Entry(searchWind)
    l1.pack(anchor = 'center')
    search_term.pack(anchor='center')
    btn = Button(searchWind, text="Search", command = lambda: search_event(btn, searchWind, api, search_term))
    btn.pack(anchor = 'center')
    return searchWind.state()
```

**Justification**

From looking through the code, there's no loops in this piece of code. Therefore, with the parameters search_form(tk, api) where tk is the tkinter constructor and api is just the google_calendar_api. Thus, search_form(tk, api) gives us 100% of the code coverage.

**Test case** - create_ui
**Tester** - Yew Yee Perng (32205481)
**Technique used** - Statement coverage
**Parameters used for checking** - create_ui(tk , api)

```
def create_ui(archive, api):
    tk = Tk()
    tk.geometry("850x1000")
    tk.title("MyEventManager")
    currentDay = datetime.now().day
    currentMonth = datetime.now().month
    currentYear = datetime.now().year
    maxdate1 = date(2050,12,31)
    cal = Calendar(tk, selectmode = 'day', year = currentYear, month = currentMonth, day = currentDay, maxdate = maxdate1)
    cal.pack(pady = 20, fill = "both", expand = True)

    btn = Button(tk, text="Create Event", command = lambda: openForm(tk, api))
    btn.pack(pady = 10)

    btn = Button(tk, text="View Event Archive", command = lambda: view_archive(tk, archive, api))
    btn.pack(pady = 10, anchor = 'center')

    btn = Button(tk, text="Import JSON", command = lambda: import_json(api))
    btn.pack(pady = 10, anchor = 'center')

    btn = Button(tk, text="Export JSON", command = lambda: export_json(api))
    btn.pack(pady = 10, anchor = 'center')

    btn = Button(tk, text="Clear Events", command = lambda: force_refresh(tk, archive, api))
    btn.pack(pady = 10, anchor = 'w')

    header = Label(tk, text = "Events: ", font='bold')
    header.pack(pady = 10, anchor = "w")

    btn = Button(tk, text="Search Events", command = lambda: search_form(tk, api))
    btn.pack(pady = 10, anchor = 'w')
```

```
    options_month = [
        "1",
        "2",
        "3",
        "4",
        "5",
        "6",
        "7",
        "8",
        "9",
        "10",
        "11",
        "12"
    ]
    clicked = StringVar()
    clicked.set("1")
    month_lbl = Label(tk, text = "Select Month: ")
    month_lbl.pack(anchor= "w")
    menu_month = OptionMenu(tk, clicked, *options_month)
    menu_month.pack(anchor = "w")

    year_lbl = Label(tk, text = "Input Year: ")
    year_lbl.pack(anchor= "w")
    temp = Entry(tk)
    temp.insert(0, "2022")
    temp.pack(anchor="w")

    btn = Button(tk, text="Get Events", command = lambda: updating_tasks(archive, tk, temp, clicked, api))
    btn.pack(pady = 10, anchor = 'w')

    tk.mainloop()
    return 0
```

**Justification**

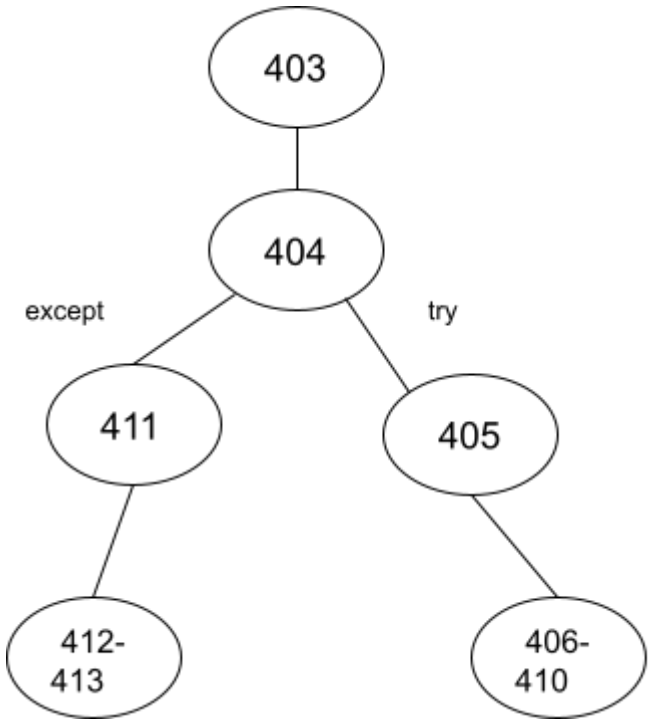
From looking through the code, we can see that there are no loops. Therefore, with the parameters create_ui(tk, api) where tk is the tkinter constructor and api is just the google_calender_api. Thus, open_form(tk, api) gives us 100% code coverage.

**Test Identification**: export_json
**Tester** - Yew Yee Perng (32205481)
**Technique used** - path coverage

Conditions - A => event_list = api.events().list(calendarId='primary').execute()

| Test | A | outcome |
|---|---|---|
| 1 | T | T |
| 2 | F | F |

**Control flow diagram (CFG)**
403 in the oval means line 403 in the MyEventManager.py code




**Justification**
Path coverage is used on this particular test case. There are multiple paths/ branches that could be led to due to different conditions. For this case, there is a try and except statement in the code which could lead to different outcomes. Therefore, path coverage is suitable to be used as it could clearly show the paths and different outcomes of different paths taken.