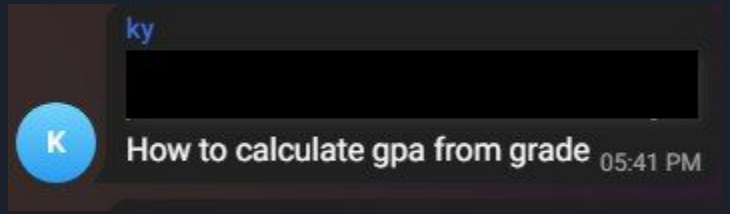
Inspiration

“How to calculate gpa from grade” was the question our classmate Kian Yew asked in the class chat, right after receiving our midterms results. The opportunity to work on a GPA Calculator to help SUTD students calculate their grades thus presented itself.

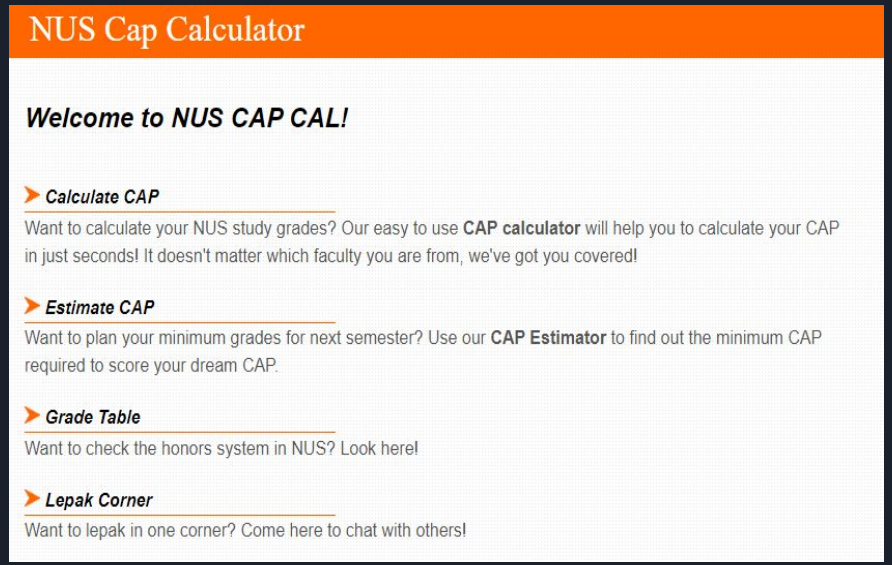
While this calculation can be done manually, it is a long and tedious process of researching individual mod components, punching in numbers on the calculator and putting our memory to the test.

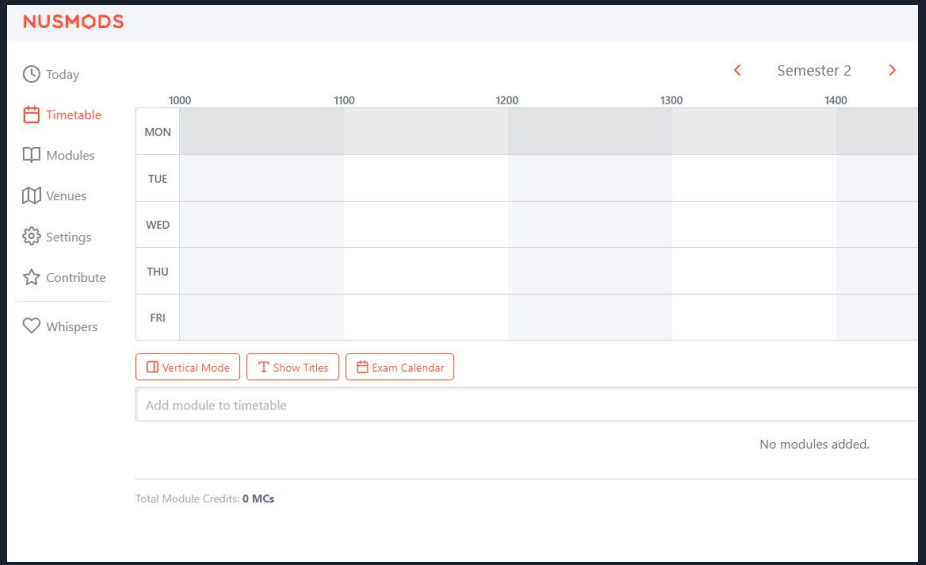
This program simplifies the process and conveniences students.



Inspiration

NUS CAP Calculators, NUS Mods (timetable) exists, so why not SUTD's very own?





Target Audience

Our programme aims to help the entire student population in SUTD.

Students are busy with projects and do not want to gather the information required to calculate their GPA for each module separately and manually.

This program provides them with the required information that is stored and can be retrieved and edited at any time. They can then concentrate on studying and allocating the limited time and resources to their weaker subjects.

Eg. As a Term 1 freshmore new to the school system, Zorye is finding school hard and does not want to go for bootcamp. Currently, he is in Week 11 of the term, and has only received grades for certain components of each module. He wants to find out how much he needs to score for the remaining components for all his individual modules and overall term so he can gauge his performance and adjust his study schedule to achieve at least 60% for the term.

Concepts we applied from lessons

String manipulation methods of splitting and stripping

```
credits_list = content.split("\n")[-1].split(": ")[1].strip("()").split(", ")
```

```
10.014 | CTD | Computational Thinking Design
Class Participation: (0.6, 2)
Visual Programming: (3.0, 9)
Assignment 1: (12.0, 15)
Assignment 2: (3.0, 20)
Python Programming: (3.0, 9)
1D: (2.0, 10)
2D: (3.0, 10)
Finals: (25.0, 25)
Credits: (6.19, 12)
```

Concepts we applied from lessons

List and string slicing with both positive and negative indices

```
106                 current_mod_credits, max_mod_credits = tuple([float(credits_list[0]), int(credits_list[1])])
```

```
225             term_mod_split = [word.lower() for word in term_mod.split(" | ")]
226             # Checks if word is Eg. 10.013 | Math | any of Maths/Modelling/Analysis
227 ▼           if mod in term_mod_split[:2] or mod in term_mod_split[-1].split(" "):
```

Concepts we applied from lessons

Dictionary methods of .get(), .keys(), .values(), .items()

```
399     class_part = grades.get("Class Participation", None)
```

```
84                          term_mods = content.get(term, None).keys()
```

```
162             term_mod = list(grades_dict.keys())[0]
163             components = list(grades_dict.values())[0].items()
```

Concepts we applied from lessons

Enumeration of both lists and dictionaries

```
for i, v in enumerate(files, start=1):
    print(f"{i}. {v.rstrip('.txt')}") if not v.startswith("_") else None
```

```
We found the following files
1. term_1_dtp
2. term_1_hass
3. term_1_math
4. term_1_ctd
6. term_1_physics
Input file to edit:
```

Concepts we applied from lessons

Enumeration of both lists and dictionaries

```
373    qns = [
374    "How often you attend classes",
375    "How often you ask or answer questions in class?",
376    "How often you submit assignments on time",
377    "How active you are during group discussions"
378    ]
379    for i, v in enumerate(qns, start=1):
380        ans = input_validated(f"{i}. {v}: ", (0, 10))[1]
381        score += ans
```

```
On a scale of 0-10, input
1. How often you attend classes: 1
2. How often you ask or answer questions in class?: 2
3. How often you submit assignments on time: 10
4. How active you are during group discussions: 5
```

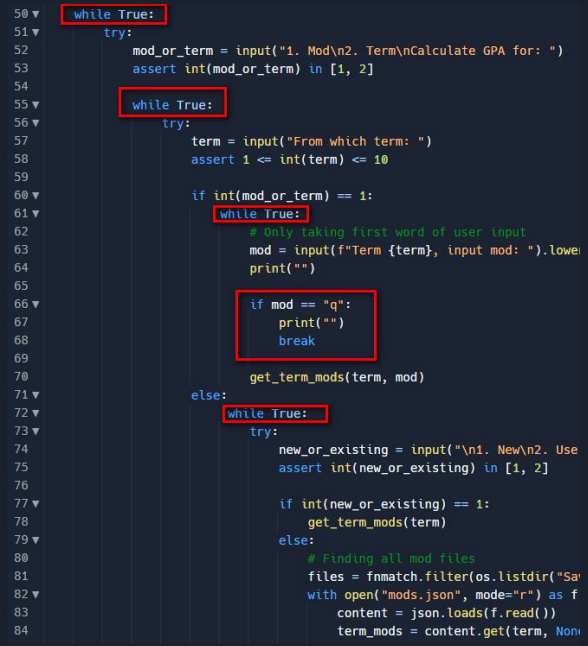
Concepts we applied from lessons

Enumeration of both lists and dictionaries

```
173 ▼                for i, (component, (grade, weightage)) in enumerate(components, start=1):
174
175 ▼                    if i == int(component_choice):
176                         actual, weightage = calculate_component(component, weightage)
177                         grades_dict[term_mod][component] = actual, weightage
178                         current_pct += actual
179 ▼                   else:
180                         current_pct += grade
```

Concepts we applied from lessons

Nested while loops with try and except blocks



```
while True:
    try:
        mod_or_term = input("1. Mod\n2. Term\nCalculate GPA for: ")
        assert int(mod_or_term) in [1, 2]

        while True:
            try:
                term = input("From which term: ")
                assert 1 <= int(term) <= 10

                if int(mod_or_term) == 1:
                    while True:
                        # Only taking first word of user input
                        mod = input(f"Term {term}, input mod: ").lowe
                        print("")

                        if mod == "q":
                            print("")
                            break

                        get_term_mods(term, mod)
                else:
                    while True:
                        try:
                            new_or_existing = input("\n1. New\n2. Use
                            assert int(new_or_existing) in [1, 2]

                            if int(new_or_existing) == 1:
                                get_term_mods(term)
                            else:
                                # Finding all mod files
                                files = fnmatch.filter(os.listdir("Sa
                                with open("mods.json", mode="r") as f
                                    content = json.loads(f.read())
                                    term_mods = content.get(term, Non
```

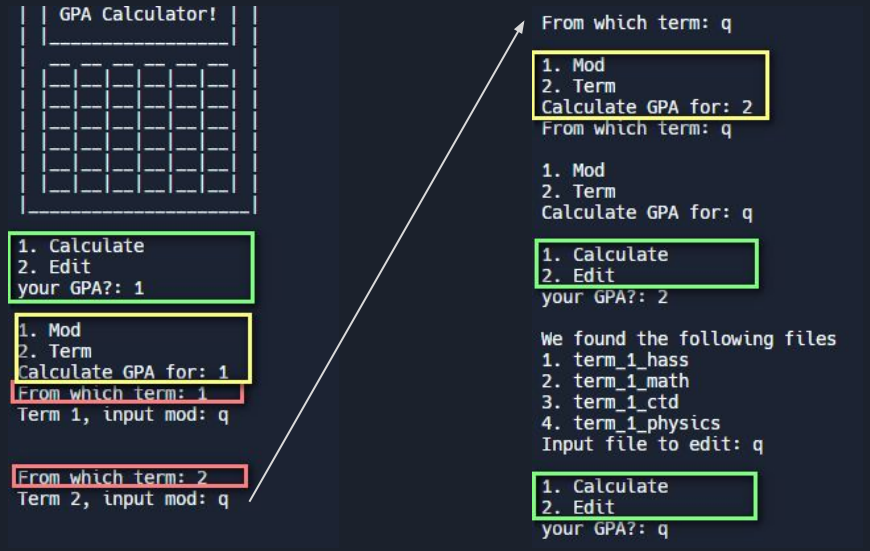
```
                        except ValueError:
                            if new_or_existing == "q":
                                print("")
                                break
                            else:
                                print("Invalid! Input only integers\n")
                        except AssertionError:
                            print("Invalid! Input 1 or 2\n")


                except ValueError:
                    if term == "q":
                        print("")
                        break
                    else:
                        print("Invalid! Input only integers\n")
                except AssertionError:
                    print("Invalid! Input 1-10\n")

    except ValueError:
        if mod_or_term == "q":
            print("")
            break
        else:
            print("Invalid! Input only integers\n")
    except AssertionError:
        print("Invalid! Input 1 or 2\n")
```

Concepts we applied from lessons

Our nested while loops and functions not always containing return statements enable user to backstep/backtrace from any point, in the event of an accidental input.



Concepts we applied from lessons

Program will not crash unexpectedly and user will not experience unpleasant errors, due to our usage of try-except blocks and validation of user-inputs that are

1. More pythonic, hence good programming practice,
2. Faster when exceptions are exceptional which is expected to be the average case as clear instructions for a proper input are given when user provides undefined inputs
3. Allow program to continue running without crashing due to an unhandled exception

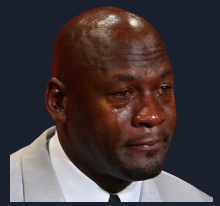
when compared to if-else

```
1. Calculate
2. Edit
your GPA?: oops

Traceback (most recent call last):
  File "main.py", line 63, in <module>
    boot_up()
  File "main.py", line 41, in boot_up
    start(choice)
  File "main.py", line 10, in start
    GPACalculator.introduce()
  File "/home/runner/CTD-1D/GPACalculator.py", line 27, in introduce
    assert int(calculate_or_edit) in [1, 2]
ValueError: invalid literal for int() with base 10: 'oops'
```

User:

After keying in all data:

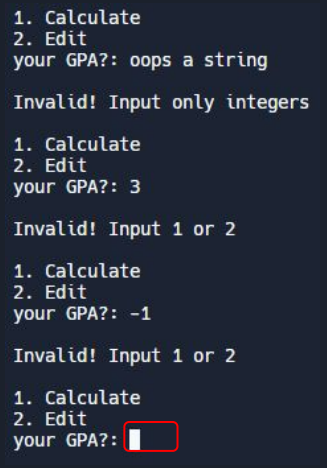


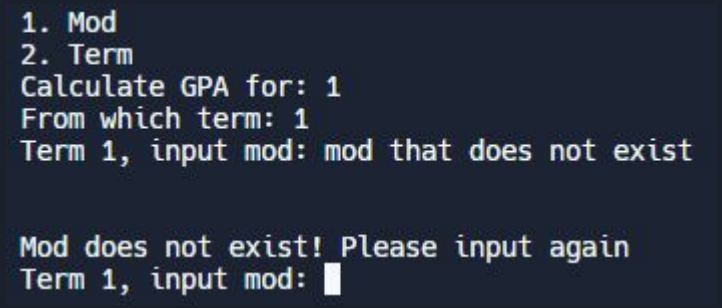
Concepts we applied from lessons

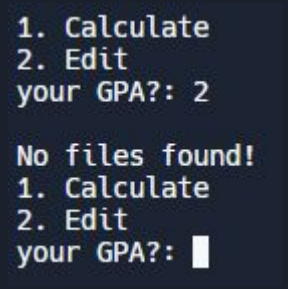
Program will not crash unexpectedly and user will not experience unpleasant errors, due to our usage of try-except blocks and validation of user-inputs that are

1. More pythonic, hence good programming practice,
2. Faster when exceptions are exceptional which is expected to be the average case as clear instructions for a proper input are given when user provides undefined inputs
3. Allow program to continue running without crashing due to an unhandled exception

when compared to if-else







Concepts we applied from lessons

Opening, reading and writing to files with various modes

```
157 ▼           with open(f"Saved_Data/{file_choice_name}", mode="r+") as file:
158                 grades_dict, credits = text_to_dict(file.read())
```

How files work - seeking, truncating and flushing

```
186                     file.seek(0)
187                     file.truncate(0)
188                     file.write(dict_to_text(grades_dict))
189                     file.flush()
```

Concepts we applied from lessons

Some usage of classes and objects
Instantiation, Creation of a class method

```
156  # Class module. Stores the respective widgets and other relevant datas
157  class Module:
158      def __init__(self, mod_entry, mod_scores, name):
159          self.entry = mod_entry
160          self.scores = mod_scores
161          self.name = name
162
163      def calculate(self):
164          # clear_output(wait=False) # Clears the console screen with ev
165          print(f"calculating {self.name}!")
```

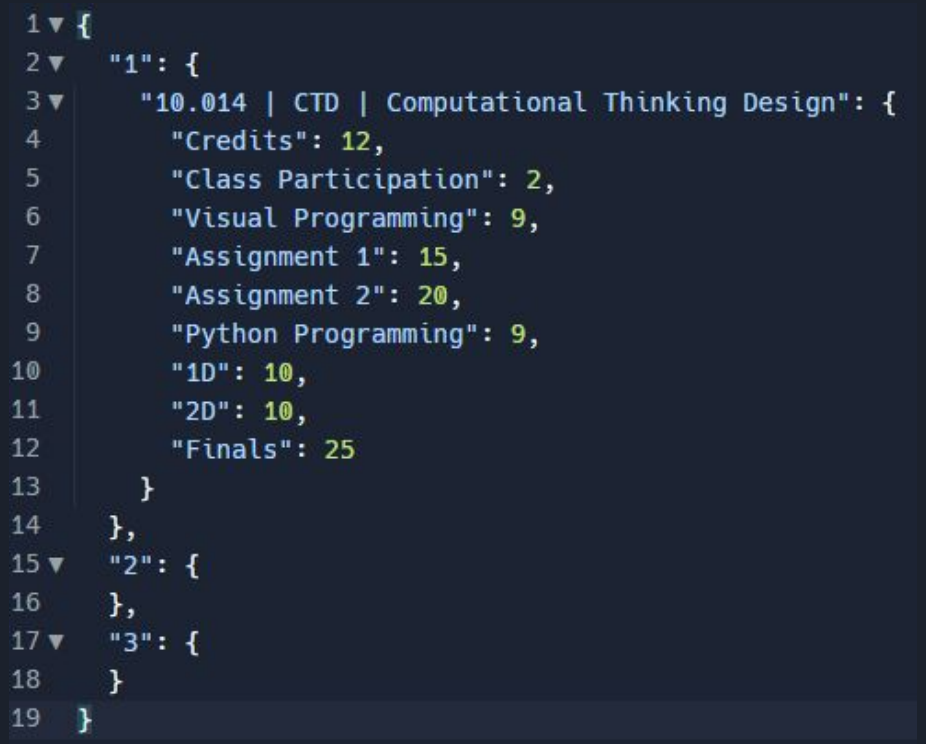
```
221  # Configures the 4 Buttons that calculates different modules
222  calc_buttons['Math'].configure(command=math.calculate)
223  calc_buttons['Physics'].configure(command=phy.calculate)
224  calc_buttons['HASS'].configure(command=hass.calculate)
225  calc_buttons['CDT'].configure(command=cdt.calculate)
226  calc_buttons['DTP 1'].configure(command=dtp.calculate)
```

Special Methods

```
250 ▼           if isinstance(actual_score, tuple):
```

Special Library and Methods

```python
with open("mods.json", mode="r") as file:
    content = json.loads(file.read())
```



```json
{
  "1": {
    "10.014 | CTD | Computational Thinking Design": {
      "Credits": 12,
      "Class Participation": 2,
      "Visual Programming": 9,
      "Assignment 1": 15,
      "Assignment 2": 20,
      "Python Programming": 9,
      "1D": 10,
      "2D": 10,
      "Finals": 25
    }
  },
  "2": {
  },
  "3": {
  }
}
```

Special Library and Methods

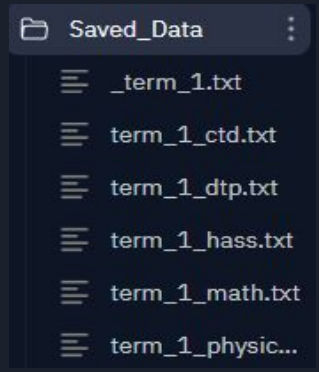
OS to clear the screen

```
289                 os.system("clear")
```

OS to search in specified directory

```
142     files = fnmatch.filter(os.listdir("Saved_Data"), "*.txt")
```

```
We found the following files
1. term_1_dtp
2. term_1_hass
3. term_1_math
4. term_1_ctd
6. term_1_physics
```



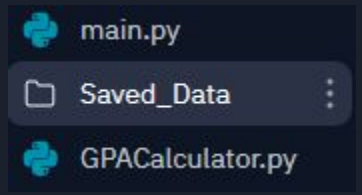
Special Library and Methods

Fnmatch internally making use of inbuilt re library to find files matching the given expression in a given directory

```
81                    files = fnmatch.filter(os.listdir("Saved_Data"), "[!_]*.txt")
```

Good Programming Practices

keeping itself self-contained and not automatically run, if imported in future




```
62 ▼ if __name__ == "__main__":
63       boot_up()
```

Helper Functions

```
350  ############################################## Helper functions ##############################################
351 ▶ def input_validated(qn: str, bounds: tuple): …
369 ▶ def give_survey(total_percentage: int) -> float: …
393 ▶ def advise(grades: dict) -> str: …
424 ▶ def dict_to_text(grades_dict: dict) -> str: …
436 ▶ def text_to_dict(grades_text: str) -> dict: …
451 ▶ def display_components(term_mod:str, components: dict, enumerated: bool): …
465 ▶ def save_data(file_name: str, grades_text: str): …
```

Good Programming Practices

Proper indentation, expected parameter types, return types, docstrings, comments

```
369 def give_survey(total_percentage: int) -> float:
370     """Gives survey to estimate Class Participation"""
371     score = 0
372     print("\nOn a scale of 0-10, input")
373     qns = [
374     "How often you attend classes",
375     "How often you ask or answer questions in class?",
376     "How often you submit assignments on time",
377     "How active you are during group discussions"
378     ]
379     for i, v in enumerate(qns, start=1):
380         ans = input_validated(f"{i}. {v}: ", (0, 10))[1]
381         score += ans
382 
383     # 32/40 = 4/5
384     fraction = round(score/(10*len(qns)), 1)
385     # 80%
386     percentage = 100*fraction
387     # 12%
388     percentage_out_of_total = round(fraction*total_percentage, 1)
389     print(f"Your class participation is {percentage}%, giving you {percentage_out_of_total}% out of a possible
    {total_percentage}%\n")
390 
391     return percentage_out_of_total
```