School of Computing

Module Code	M30299
Module Title(s)	Programming
Module Coordinator Other lecturers	Matthew Poole Nadim Bakhshov (responsible for this CW) Xia Han Mani Ghahremani
Assessment Item number	Item 6
Assessment Title	Python Assignment — Smart Home
Date Issued	12 Feb 2024

Schedule and Deliverables

Deliverable	Value	Format	Deadline Date	Late/ECF deadline
Program	100% — all marks for your program will be awarded in the demo	Zip file containing your code, named after your UP number (e.g., **2123456.zip**)	16:00, 11 Mar 2024	16:00, 25 Mar 2024
Demo			In your practical class between 12 and 15 Mar 2024	All late demos are to be completed by 26 Apr 2024

Notes and Advice

- The Extenuating Circumstances procedure is there to support you if you have had any circumstances (problems) that have been serious or significant enough to prevent you from attending, completing or submitting an assessment on time. If you complete an Extenuating Circumstances Form (ECF) for this assessment, it is important that you use the correct module code, item number and deadline (not the late deadline) given above.
- ASDAC are available to any students who disclose a disability or require additional support for their academic studies, with a good set of resources on the ASDAC Moodle site
- The University takes any form of academic misconduct (such as plagiarism or cheating) seriously, so please make sure your work is your own. Please ensure you adhere to our Code of Student Behaviour.
- Any material included in your coursework should be TECFAC 08 Plagiarism fully cited and referenced in APA 7 format. Detailed advice on referencing is available from the library.
- Any material submitted that does not meet format or submission guidelines, or falls outside the submission deadline could be subject to a cap on your overall result or disqualification entirely.
- Instead of referring to external sites for help, join our Module Discord channel to post your questions.
- If you need additional assistance, ask your personal tutor or the student engagement officer at ana.baker@port.ac.uk. The faculty academic tutors are also available for you to book a one-to-one session with them using the Moodle page for faculty academic tutors.
- If you are concerned about your mental well-being, please contact our Well-being service.

M302399, Item 6

Python Coursework 2: Smart Home

Scenario

A fictional UK-Finnish company, Alikoti, develops eco-friendly smart home technologies. Since 2011 they have developed fully integrated Smart Home Solutions. Since 2022 they have begun selling standalone smart home devices, including smart lights and smart fridges, to be supported by third-party software. Last year, the company made the decision to build its own Smart Home App. They have asked you to develop a prototype GUI app. This prototype must provide a graphical user interface that interacts with a smart home object. The smart home object is responsible for managing data for all smart devices. The company will use this prototype to make a business case for a fully developed application.

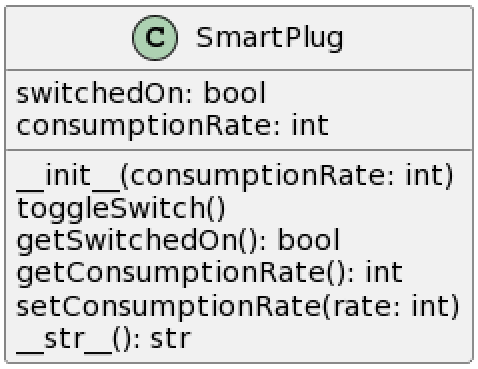
Tasks

This document contains a list of tasks. Tasks 1, 2 and 3 must be completed in a file named `backend.py` and tasks 4 and 5 should be done in `frontend.py`. It is suggested that you complete the tasks in this order. Note that it is not compulsory to complete the challenge features. Only attempt these if you feel confident and have completed tasks 1–5.

Once you are ready to submit, zip (compress) the folder containing both files (`backend.py` and `frontend.py`). No other files should be included in the folder. Please make sure you submit a `.zip` file (not `.rar`, `.7z`, `.tar or .pdf`). Submit your zip file using the link on Moodle (in the Assessments tab) called "Item 6 — Python Smart Home Assignment Dropbox".

Task 1 — `SmartPlug` Class [15 marks]

In `backend.py`, create a class called `SmartPlug`. This class represents a smart plug object with the instance variables and methods described below:



The `SmartPlug` class should have two instance variables: `switchedOn` and `consumptionRate`. `switchedOn` represents the plug's operational status, while `consumptionRate` indicates the rate of consumption within the range of 0 to 150. The constructor should initialise `switchedOn` to `False` (representing off) and set `consumptionRate` to an initial value set from the supplied parameter. The `SmartPlug` class should include a `toggleSwitch` method to modify the operational status (from off to on, and on to off), an accessor method for retrieving the operational status, and both accessor and mutator methods for handling the consumption rate. Make sure that your accessor and mutator methods handle invalid parameters appropriately. Finally, add a `__str__` method to provide a human-readable string representation of a `SmartPlug` object.

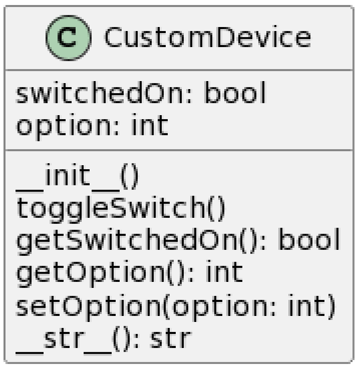
Testing

Test your `SmartPlug` class by writing a `testSmartPlug` function outside the `SmartPlug` class. The `testSmartPlug` function should:

1. Create an instance of the `SmartPlug` class with a consumption rate of 45.
2. Toggle the status of this plug by calling its `toggleSwitch` method.
3. Print the value of `switchedOn` using the appropriate accessor method.
4. Print the value of `consumptionRate`, set it to a new valid value (of your choice), and then print it again.
5. Print the `SmartPlug` object.

Task 2 — CustomDevice Class [15 marks]

Start by looking at table 1 to find out which device you need to create a class for. You should add this class and its test function to backend.py. The diagram and description below are for a generic CustomDevice class that demonstrates the basic idea of what you will have to do from table 1. The class name and option instance variable should be replaced by the class name and instance variable for your custom class from table 1.



The CustomDevice class should have two instance variables: switchedOn, reflecting the switch state of the CustomDevice, and option, representing the CustomDevice's specific option as outlined in table 1. The constructor should initialise switchedOn to a default state of off and set option to its default value from table 1. The class should have a method for toggling the switch state and an accessor method for retrieving it. The class should also have both accessor and mutator methods for handling the CustomDevice's option. Also, ensure your mutator method handles invalid parameters. When an object of your CustomDevice is printed, it should display information about its switch state and the current value of the option (so implement the `__str__` method appropriately).

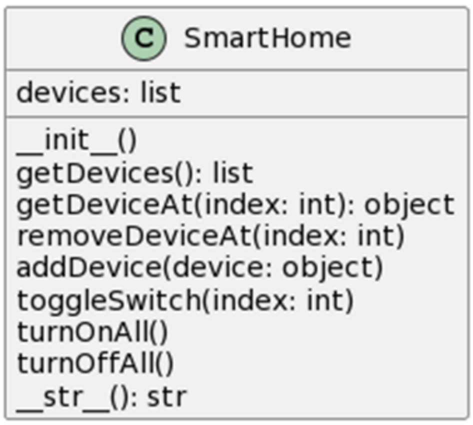
Testing

Test your device by writing a testDevice function (this should be named to reflect your particular custom device), which should:

1. Create an instance of your CustomDevice class.
2. Toggle the status of this device using the toggleSwitch method.
3. Print the switchedOn instance variable using the appropriate accessor method.
4. Print the current value of the option instance variable (from table 1). Then set it to a new value (of your choice). Next, print it again.
5. Print the CustomDevice object.

Task 3 — `SmartHome` Class [15 marks]

In `backend.py`, add a `SmartHome` class that manages and controls a collection of devices.



The `SmartHome` class should include a single instance variable, `devices`, representing a collection of devices. Each element of this collection can either be a `SmartPlug` or a custom device. The constructor should ensure that `devices` is initially empty. The class should have methods that let the user access all `devices`, retrieve or delete a specific device, and add a device. The user should be able to toggle a specific device but also turn all of them on or off at once. Additionally, when a smart home is printed, it should display a well-formatted message describing all of its devices and their attributes. Ensure that your accessor and mutator methods handle invalid parameters.

Testing

Test your class by writing a `testSmartHome` function, which should:

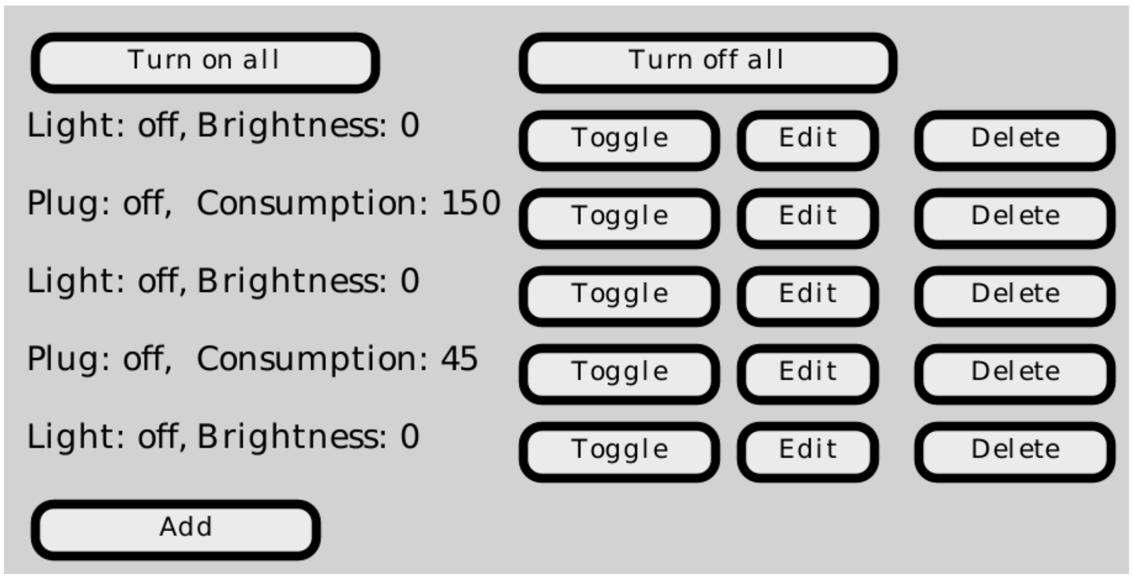
1. Create an instance of the `SmartHome` class and two instances of the `SmartPlug` class with consumption rates of 45. Additionally, create an instance of your custom device.
2. Toggle the first plug, hence turning it on. Then set its consumption rate to 150. Then, set the `consumptionRate` of the second plug to 25. Lastly, set the option of the custom device to a value of your choice.
3. Add both plugs and the custom device to the smart home object.
4. Toggle the status of the second plug using the appropriate method of the smart home object.
5. Print the smart home object.
6. Turn on all devices in the smart home and print the smart home object again.
7. Remove the first device and print the smart home.

Task 4 — `SmartHomeSystem` GUI [15 marks]

In a file called `frontend.py`, import the class(es) that you have defined in `backend.py`. In a `setUpHome` function, create an instance of the `SmartHome` object. Allow the user to populate it with five devices of their choosing using the shell (i.e. the function needs to give appropriate prompts to the user). The user should be able to specify the consumption rate for each smart plug added.

Create a `SmartHomeSystem` class for the GUI of your smart home system that should look like the figure below, but with appropriate fonts, sizes, colours, and spacing to make the interface look pleasant. Note that at this stage, you only need to display the widgets for your GUI (none of the buttons needs to work for this task). Then, in a separate `main` function, create a `SmartHomeSystem` object and populate it with the `SmartHome` object created in `setUpHome`.

The example below assumes a student number ending with 0 (for example, “up2123410”). The user has also created three `SmartLights` and two `SmartPlugs` (with consumption rates of 150 and 45) at the beginning of program execution before the GUI is displayed:



Turn on all
Turn off all
Light: off, Brightness: 0
Toggle
Edit
Delete
Plug: off, Consumption: 150
Toggle
Edit
Delete
Light: off, Brightness: 0
Toggle
Edit
Delete
Plug: off, Consumption: 45
Toggle
Edit
Delete
Light: off, Brightness: 0
Toggle
Edit
Delete
Add

Task 5 — GUI interactivity [15 marks]

Once you have successfully built your GUI, you should implement its functionality. For each of the buttons on the GUI, you need to write the code in the `SmartHomeSystem` class that implements the functionality of the button. For instance:

- When the user clicks on the “Turn on all” button, all devices should be turned on: the backend needs to update, and the GUI should reflect this change. The same goes for the “Turn off all” button.
- When a device's “Toggle” button is clicked, the details of the device should be updated. Clicking on an “Edit” button should result in a new window where the user can modify all attributes of that device. Finally, the “Delete” button should remove the device. Clicking on any of these buttons should result in both the backend and frontend being updated.
- When the user clicks on the “Add” button, a new window should appear where they are given the option of whether they want a smart plug or a custom smart device. For new smart plugs, the user should be able to choose a consumption rate. The device should then be created and added to the backend (and GUI).

Challenge features [23 marks]

Make sure to only attempt these challenges once you have completed all the above tasks. To ensure that your solution to the challenges does not impact your answer to tasks 1-5, copy your existing files and paste them in `backendChallenge.py` and `frontendChallenge.py`. Then make your changes to these copies. You are free to add as many of the following features/enhancements to your system. The following tasks are of varied difficulty. Refer to the documentation pages of Tkinter for challenges 2-6.

1. **Inheritance** — Improve the code quality of the backend by making appropriate usage of inheritance. **[4 marks]**

2. **Advanced Tkinter Widgets** — Tkinter provides other widgets such as `CheckButton` or `SpinBox` which we have not covered in the classes. Currently, the user can only toggle each device straight from the main window. Modify your GUI to allow the user to modify other attributes of the devices directly from the main window using the widgets of your choice. You may also want to use similar widgets in the secondary windows of your app. **[4 marks]**

3. **Custom Device Visualisation** — Enhance the GUI to represent each plug/device visually based on its type. For example, display a graphic or icon that represents your plugs/devices. This visual representation should provide an intuitive way for users to identify and interact with plugs/devices. Hint: look at the `Photoimage` widget. **[3 marks]**

4. **Interface & Accessibility setting** — Improve the accessibility of your app by providing an "Accessibility settings" button. This needs to open a new window offering similar functionality to Moodle's "Interface & Accessibility settings" menu. The user should be able to alter text size, change between light and dark mode, and also define a custom colour scheme (consisting of a background and text colour). Hint: Visit the `tkColorChooser` module. **[4 marks]**

5. **Permanent Data Storage** — Currently, the user needs to create the devices one by one in the shell and the smart home system does not have permanent storage (i.e., upon closing the smart home system, all data will be lost). Investigate the `tkFileDialog` module of Tkinter which allows the user to upload a file (e.g., a text or a CSV file) where each row represents a device. Your app should also allow the user to save the state of a smart home to a file. **[4 marks]**

6. **Device Scheduler** — Add a clock with hour values 0-23 which increment every 3 seconds to simulate the passing of time; "minutes" do not need to be implemented or displayed. Include a scheduling feature for devices using this clock feature, allowing users to set specific times for devices to turn on or off automatically. The main GUI window should show the clock and users should be able to see the devices turn on and off at the set times. **[4 marks]**

Table 1

Using the **final digit** of your student number, you are required to build a custom smart device. The table below shows what the name of the class should be and the option that it needs to have.

Final Digit	Custom Smart Device	option	default value
0	`SmartLight`	`brightness` (as a percentage, i.e., 0-100)	0
1	`SmartFridge`	`temperature` (1, 3, or 5 degrees Celsius)	3
2	`SmartHeater`	`setting` (0–5)	0
3	`SmartTV`	`channel` (1-734)	1
4	`SmartSpeaker`	`streaming` ("Amazon", "Apple" or "Spotify")	"Amazon"
5	`SmartDoorBell`	`sleepMode` (True or False)	False
6	`SmartOven`	`temperature` (0–260 degrees Celsius)	0
7	`SmartWashingMachine`	`washMode` ("Daily wash", "Quick wash" or "Eco")	"Daily wash"
8	`SmartDoor`	`locked` (True or False)	True
9	`SmartAirFryer`	`cookingMode` ("Healthy", "Defrost" or "Crispy")	"Healthy"

Demonstration & Mark Allocation

You need to **demonstrate your program** to a member of staff in your Programming **practical session, timetabled in the period 12th—15th March**. We will execute your submitted program, and we will ask you question(s) about how you wrote it and how it works.

All the marks for the assignment will be awarded during the demonstration, so you must attend: **failure to attend the demonstration will result in your work being recorded as a non-submission**, and demonstrating your program late may result in your mark for the assignment being capped at 40%. **Late demonstrations must be done by the 26th of April,** or your work will be recorded as a non-submission. You need to ask Nadim Bakhshov, Matthew Poole, Xia Han or Mani Ghahremani at the beginning of a drop-in (Friday at 9am) or a practical class if you wish to do a late demo.

Formal written feedback and your assignment mark will be sent to you via email immediately after your demonstration has been completed. If you do not receive this email, then your mark may not have been recorded, and it is your responsibility to inform Nadim Bakhshov within 24 hours of the demonstration if this happens.

Marking

Each task will individually be marked for functionality and code quality. The total marks for each task are given in the heading of each task (the maximum mark is 100).

- The functionality of tasks 1 and 2 will be verified through the execution of the test functions written in the backend file.
- Tasks 3 and 4 will be verified through the execution of the GUI.

Explanation of program features [2 marks]

We will award up to 2 marks for your responses to the question(s) we ask you in the demo, independently of the other marks your submission receives. You will receive:

- 2 marks if you give clear/correct responses to the questions;
- 1 mark if you give less than clear/correct responses;
- 0 marks if your responses are poor and lead the marker to suspect that you did not

write the complete program yourself; in such an event, your work will be investigated for academic misconduct after the demonstration

General advice

Above all, begin the task early and avoid postponing its completion until the last minute. Your work will almost always suffer if you leave it until too late. Furthermore, technical problems are likely to be overcome if encountered early and do not usually constitute an acceptable reason for lateness. If you find the task very difficult, remember that you do not have to provide a complete solution to achieve a pass mark.

Support

The last few practicals before the deadline can be used to help with the coursework. Simon Jones and Eleni Noussi are also able to give advice on the coursework via one-to-one support sessions. Alternatively, join our Discord channel and post your questions.

Other queries should be addressed to Nadim Bakhshov by email. Any reported errors on this handout, or other common issues, will be communicated via the Python Smart Home Coursework Frequently Asked Questions document in the General section on Moodle, so please check there before asking a question.

You are allowed to ask others for limited help, but you must write your own code. You should not, therefore, make extensive use of AI assistance (ChatGPT, Bard, Copilot etc.). Remember that anything you submit must be your own work, and you will be asked in the demonstration to explain parts of your code.

Important

This is an individual coursework, and so the work you submit for assessment must be your own. Submission of work that is not your own, or unfair collaboration, is plagiarism, which is a serious academic offence. Any suspected cases of plagiarism will be dealt with in accordance with University regulations.

Nadim Bakhshov
12 February 2024