|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Task:** | | **3** | | |
| **Task Title:** | | **Portfolio** | | |
| **Task Code:** | | **AT2 POR-Task-3** | | |
|  | |  | | |
| Assessment type (): | | | | |
|  | Questioning (Oral/Written) | |  | Portfolio |
|  | Practical Demonstration | |  | Project |
|  | 3rd Party Report | |  | Other – Please Specify |

|  |
| --- |
| The base requirements this assessment task include:   * Web server, Python interpreter and database server * IDE or editor for developing Python programs (only PyCharm supported by the college) * Raspberry Pi with SenseHat * Access to Office 365 & Microsoft Word * Report Template (Portfolio: Part 3 Document Template) as supplied   Use of some of these items may not occur in this part of the assessment task. |
| Assessment Due This assessment is split into components that have several due dates:   * Week 07 17:00 (5:00PM) on the day of the scheduled lecture. * Week 09 17:00 (5:00PM) on the day of the scheduled lecture. * Week 12 17:00 (5:00PM) on the day of the scheduled lecture. * Week 14 17:00 (5:00PM) on the day of the scheduled lecture.   Refer to Blackboard for most accurate dates, which may alter due to unforeseen circumstances.  We also will endeavour to update these document(s) at the same time. |
| Instructions Follow the steps listed in this assessment item. Please note that **additional** information may be given.  Submission of the documentation, code, and associated items is at the end of each part of the portfolio.  Each part of the portfolio has a deadline for submission.  It is advantageous to you to attempt to meet the deadline provided. |
| Important If you are using a different configuration of tools and equipment for this assessment item, then assistance in this and subsequent parts of the portfolio to ensure the systems work correctly will be limited. |
| Scenario / background In this Portfolio Task, we’re going to have a closer look at the four pillars of OO:   1. Abstraction 2. Polymorphism 3. Inheritance 4. Encapsulation |
| General Instructions We provide a document template for your answers.  Save the file as:   * XXX-IoT-Port-Part-3.docx   Replacing the XXX with your initials.  For example, Adrian Gould would use AG-IoT-Port-Part-3.docx for his submitted filename.  Upload any code as a PyCharm project in a zip-file. Remove the virtual environment (**venv** or **.venv**) from the zip-file before uploading it to Blackboard. |
| Answering Questions When a step includes a question, you must attempt to answer it.  There is a minimum and maximum number of words to use for each answer.  If a step has more than one question, these maxima and minima are a total for all the questions in that specific step.  All answers must be in complete sentences unless indicated. |
| Sources of Information In industry, it is good practice to keep track of where information was obtained. This is especially true if it is a written document, or even code.  If you answer any questions using information from web sites, please include the site name and URL (Web site address) after the answer. Likewise, include the title and author for books and magazine articles. For example:   * RS Electronics Ltd: <https://au.rs-online.com/> * Slack API Documentation, Users List Method: <https://api.slack.com/methods/users.list>  Code Storage We advise that you create a GIT repository on GitHub and use this to store a copy of your work.  You may also use OneDrive within your college Office365 to store a backup of your code or keep a copy on a USB thumb drive. |
| *This space left intentionally blank.* |

| **STEP** | **Task to perform** | Words Min/Max |
| --- | --- | --- |
| 00 | Create Evidence Document Make sure you have followed the instructions on creating the answer document, as given in the General Instructions.  Familiarise yourself with the content and document your progress in this assessment.  Make sure that you complete the title page of the document.  At any stage during this assignment, you may consult the stakeholder(s) or their representative(s). |  |
|  | *This space left intentionally blank.* |  |
| 01 | Download the “Smileys” project You can download the Smileys project from Blackboard (it is contained in a zip-file).  Download the zip-file onto your Raspberry Pi and extract it in a suitable location.  After extraction, the directory structure should look something like this:  Table  Description automatically generated  The **main.py** script is the entry point for the project. You should be able to run the script on your Raspberry Pi by typing **python3 main.py** in a terminal. Make sure to **cd** into your project directory (this means Change Directory).  You may ignore the **requirements.txt** for now. The file **README.md** contains a brief description of what the project is about. It is good practice to add a README to any project you create.  Take a screenshot of the result of the following two commands and put it in the answer document ($ is the prompt; don’t type that):  **$ ls** list files of my smileys project folder atatched **$ python3 main.py =🡺** using trinket, so screenshot attached | n/a |
|  |  |  |
| 02 | Find the classes The project contains a few classes. Some are “base classes” (sometimes called super classes) and some are “subclasses”, which are derived from the base classes.  Answer the following questions:   1. How many classes can you identify in the project?   Smiley, Sad, Happy, Blinkable, ABC, time, main, Sensehat   1. In your own words, describe how ‘abstraction’ is visible in this project. When we import a class and call a function from it we get it to run without knowing how that class works it is “hidden” from us, and in most cases we don’t really need to know how it works, we just want to use it 2. Describe which of these classes are subclasses and which are base (or super) classes. Smiley, Sad are Base classes, ABC is an abstract base class, blinkable is its subclass an interface, Happy is a subclass of both Smiley and Blinkable, Sensehat is a base class and Time is a base class 3. What is the name of the process of deriving from base classes?   Inheritance | n/a |
|  | *This space left intentionally blank* |  |
| 03 | Not all classes are the same Investigate the classes **Happy** and **Sad**.  Notice the similarities and the difference. Be careful, what may look like a similarity may actually contain a subtle but important difference.  In your own words, describe:   * The commonalities between **Happy** and **Sad** * Both use draw\_mouth and draw eyes functions and have an \_\_init\_\_ constructor * The differences between **Happy** and **Sad** * Sad doesn’t have a blink function (which is derived from the blinkable class, Happy is a subclass of Smiley and of Blinkable. Whilst Sad is a subclass only of Smiley * Is there anything that stands out as a difference? Happy can use the abstract blink method as it inherits from Blinkable, but Sad cannot | n/a |
|  | *This space left intentionally blank.* |  |
| 04 | Where’s the Sense(Hat) in that? You will have noticed by now that the project uses the SenseHat to display the smileys on the display.  Yet, in **main.py** there is no mention of any SenseHat at all.  Answer the following questions:   1. Where is **SenseHat** used (in which **class**)? We import Sensehat into Smiley and subsequently its use is inherited by other classes that inherit from Smiley 2. Which functionalities of **SenseHat** are used? We use the low\_light and set.pixels methods 3. What is the process of storing and potentially hiding objects in classes called? Encapsulation 4. Draw a simple Class Diagram that shows the class **SenseHat** and the containing class (the answer from a.). Make sure to use the correct relationship and multiplicities. | n/a |
|  |  |  |
| 05 | Sad smileys can’t blink (or can they?) Unlike the **Happy** smiley, the current implementation of the **Sad** smiley does not have the ability to blink.  Investigate the **Happy** smiley to see how blinking has been implemented by looking at the method **blink()**, which takes in one argument. The argument determines how long the blink lasts.  You will now add the ability to blink to the **Sad** smiley.  Follow the instructions (and answer any questions):   1. In your own words, how does the **blink()**method make the smiley blink?it calls on the draw\_eyes method passing false for wide\_open which means which we draw blank then it creates a small delay and calls the draw\_eyes method again passing in true which draws in yellow to create a blink effect ie off.delay..off 2. Create a new method called blink in the **Sad** class and ensure you use the same prototype (prototype = name + arguments):  **def** blink(**self**, delay=0.25): **"""**   **Make the sad smiley blink once with a certain delay (in s).**  **This is the implementation of the abstract method from the Blinkable abstract class.**  **:param delay: Delay in seconds**  **"""**  **self.draw\_eyes(wide\_open=False)**  **self.show()**  **time.sleep(delay)**   1. Implement the code that makes the smiley blink. You may use the implementation from **Happy** as guidance. 2. Test the code on your Raspberry Pi and watch the sad smiley blink its eyes. (You may have to adjust the **main.py** script for this.) |  |
|  |  |  |
| 06 | If it walks like a duck… If you followed the instructions in the previous question closely, you didn’t use the class **Blinkable** to make **Sad** blink. (If you did, that’s fine. In that case, however, you should assume you did not when answering the following questions.)  You did not have to use **Blinkable** to allow the **Sad** smiley to blink.  Answer the following questions:   1. What kind of class is **Blinkable**? Look at its super class for a hint.it is an abstract base class 2. Any class that uses **Blinkable** is said to “implement” it. What is another (generic) name for a class like **Blinkable**, which may be implemented by other classes? An Interface 3. What is the answer from the previous question an example of? Choose from: Abstraction, Polymorphism, Inheritance, and Encapsulation. inheritance 4. Why were you able to use it like the one defined in **Happy**, yet without using **Blinkable**? We create the Happy class and are able to instantiate an object which derives from the interface Blinkable and we implement the abstract method which was derived from it 5. Regarding the answer to d., what is this called and why does it work in Python but not in some other languages, like C#? (A hint is in the title of this question.)we are implementing an interface. Python doesn’t require a class implementing an interface to define all the abstract methods of the abstract class c# and Java do |  |
|  |  |  |
| 07 | Does a smiley have to be yellow? You don’t have to answer the above question, because obviously it doesn’t. But keep reading.  Smileys that do not feel too well are often green, while angry smileys are usually red or orange.  The current implementation only allows for yellow smileys, though. That means we’re a bit limited as to what emotions we can express with our SenseHat Smileys.  Answer the following questions:   1. Which colours are **defined** and where? White, green, red, yellow and “blank” they are defined in the Smiley class 2. What is the name of the **type of variables** that hold the colours? CONSTANTS 3. Where are the colour variables actually **used**? They are used in the \_\_Init\_\_(self) self.pixels array to define the colours on the sensehat 4. What would be an easy, albeit rather naive, way to change the colour of the smileys, for example, to green? Change the Y = self.YELLOW to Y = self.GREEN |  |
|  | *This space left intentionally blank.* |  |
| 08 | Flexible colours – Step 1 Changing the colour of the smileys once is easy. But it’s not very flexible, is it?  To allow for smileys to have different colours, we best not hardcode any values in any class, which you discovered in question 7c.  Some classes have a built-in assumption about the colour that is used. Let’s remove that assumption one step at a time.  Follow these instructions:   1. Added an instance method called **complexion** to the class **Smiley**. You can make it return **self.YELLOW**.  (Technically speaking, smileys don’t have a complexion as they have no skin, but it sounds a bit nicer than just “colour”.) 2. Update the subclasses that use the colour variable directly to, instead, use the new method **complexion**. 3. Which of the four pillars of OO have we used here: Abstraction, Polymorphism, Inheritance, or Encapsulation? (If you feel more than one applies, pick the one that applies **best**.) encapsulation 4. Check that the new code works correctly. It should still just show a yellow smiley… |  |
|  |  |  |
| 09 | Flexible colours – Step 2 The subclasses no longer make any assumptions about the colour (complexion) of the smiley. Instead, it simply “asks” the superclass by calling the new method we defined.  While that still does not provide the flexibility we crave, it will make it easier to provide this flexibility.  We will now update the base class to allow flexible colours.  Follow these instructions:   1. Add a default argument to the magic method **\_\_init\_\_()** of the **Smiley** class and call it **complexion**. Assign **YELLOW** to it. (Yes, this is allowed.) 2. Create a new **instance variable** called **my\_complexion** and assign **complexion** to it. 3. Why do we need to call this instance variable **my\_complexion** and can’t just use **complexion**? (You may refer to Question 8.) 4. Also assign **complexion** to **Y**. (Okay, not the best name, but because the scope is so small, we’ll allow it for now.) 5. Finally, update the **method** **complexion** and have it return **self.my\_complexion**. 6. Run the program and see that it still just creates a yellow smiley… |  |
|  |  |  |
| 10 | Flexible colours – Step 3 We have updated our code to allow for smileys to have different colours. The colours that are provided for us are Yellow, Green, Red, and Blue.  Let’s update the sad smiley to be Blue. (See what we did there?)  Follow these instructions:   1. Locate the initialiser method of the class **Sad** and within it the call to its super class’s initialiser method. 2. Change the call as follows:  **super().\_\_init\_\_(complexion=self.BLUE)** 3. Run code to show that the sad smiley is now blue. 4. Run code to show that the happy smiley is still yellow.   In your own words, **describe** how you would create a new angry smiley that has a Red complexion and angry looking eyes. (You don’t have to write this code, but you may do this as an exercise.) I would make create a new file similar to sad and modify this to initialise with super().\_\_init\_\_(complexion=self.R*ED*), and then modify the draw\_eyes method to be eyes = [9, 14, 18, 21] which would angle the eyes diagonally to make it look angry |  |
|  |  |  |
|  | Submission of Portfolio Work To submit the portfolio, do the following:   * Save the document with your answers as a MS Word file (.docx). * Save the code that you have produced or changes in a zip-file. Make sure to remove any virtual environment (.venv) and project directories (.idea) from the zip-file before uploading. * Open Blackboard, and locate the AT2 Portfolio Task 3 assessment * Open the assessment and upload the original word-processed document and the zip-file. * Click submit.   Whilst there is no need to use any other word processing software as you have access to Office 365 for free as a student, if you use Apple Pages, or Open Office, we will then require you to upload the original file **AND** a PDF version. |  |

# Appendix A: Code Style Guidelines

Your code will follow the PEP 8 standard.

Readability Counts  
- Zen of Python

Explicit is better than implicit.  
- Zen of Python

Other code standards available in the Presentation, “Python Coding Standards for North Metropolitan TAFE”.