Track2-PythonApprentice

**Solution: File Handling**

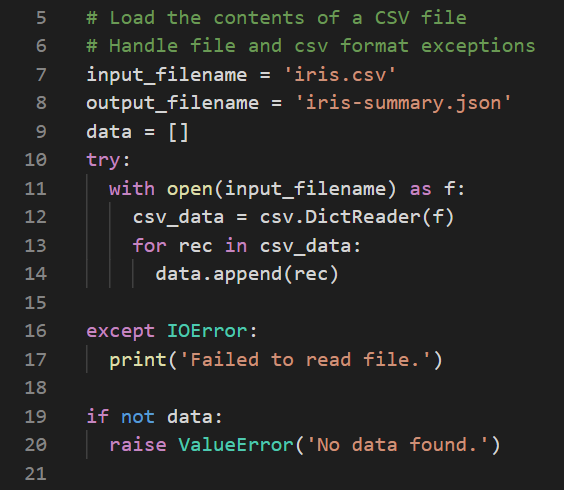
**Task 1 + 2**

On Line 1 start by importing csv, import json and import sys. Then we can use libraries like pandas or NumPy to import and export data in CSV and JSON format, but it also exists in the Python standard library. Use the standard library tools for CSV importing and JSON exporting.

[Text

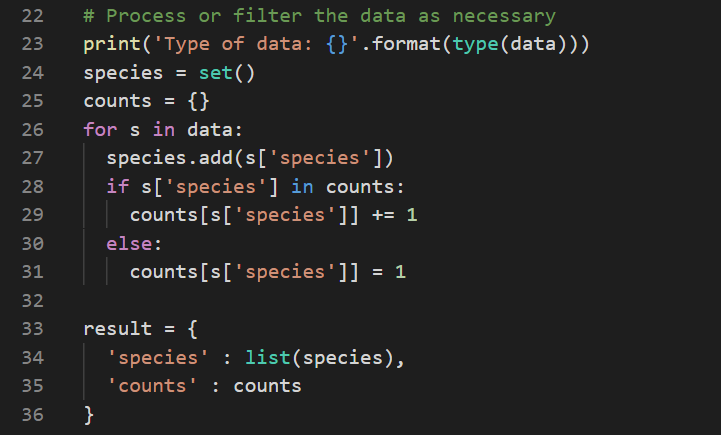
Description automatically generated](https://cdn.filestackcontent.com/d3R09JGyQ5WqBaf2fFTJ)

1. Now, the first step in the exercise is to load the contents of the CSV file, and handle file and CSV format exceptions. On Line 7, I have input file name, which is equal to the string Iris.csv. This is the file name, and it's in the local path for this project, so it will load the iris.csv. Look at the data that we're going to be importing. It's a comma separated values file with slength, swidth. For sepal length, sepal width, plength and pwidth and species. The plength and pwidth are petal length and petal width and the species is the species of Iris. We have some measurements and a species of Iris in this dataset.
2. The first step is to import this data. We'll go back to the code and on Line 9,create a list called data, which is going to hold the CSV data. data = to create an empty list. Now since we need to handle exceptions, I start with a try block on Line 10. And then inside the try block we have with open input file name as f.f will be a file handle. And then on line 12 inside the with block I have csv\_data = csv.DictReader, and pass f to this function. This uses a dictionary reader for CSV. We can step through the CSV file line by line looking at each record.
3. On Line 13, we have for rec in csv\_data, and call data.append, we're appending the record to the list that I've created. And then we have an except IOError.If there's an IOError and input output error, then we failed to read the file or failed to parse the CSV data. then print(‘failed to read file’). Then on Line 19, I check to see if data has been populated. If there is no data in our CSV file, then data will be empty. So if no data, raise a value error saying no data found, and this will cause our program to abort.

[](https://cdn.filestackcontent.com/DEbvIzVQ1C58fZc9vk3L)

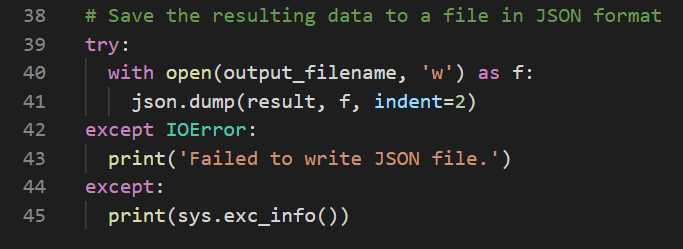
**Task 3**

1. The next step is to process or filter the data as necessary. Just for the sake of interest, print the type of data. Call a type of data and we call the type function on our data variable, just to make sure that we're working with a list. And now we’re going to count the number of species of each type in the dataset. There are three types of species of Iris in this dataset. We're going to count how many of each. We said species = set(), so it creates an empty set. And then we have a dictionary called counts, and we set that to the opening closed brace, which creates an empty dictionary.
2. Then for s in data, so data is our list of records that we've loaded from the CSV file. I call species.add with s index that is species. Species will be a set of unique values of species. And then if the s at species is in counts, so if there's a dictionary entry for the species, we increment the count by 1. So counts index that s, which is index that species plus equals 1. Otherwise, the species is not in there yet. So we add it, so we set counts at index species to 1. So for the first one, it gets populated if it doesn't exist. Now, we exit the loop.
3. On Line 33, create the results that we're going to use to output to the JSON format. Set result equals the open and close brace. So we're creating a dictionary with two keys, species and counts. Species is set to the list of species. Now, the tool that serializes our result dictionary into a JSON format does not like a set format, so it doesn't know how to handle set. So instead of using a set as species, we cast it or convert it into a list. And the counts is just a dictionary, so we used to set counts to counts.

[](https://cdn.filestackcontent.com/nPWE9ycUTWKcJ291QWOf)

**Task 4**

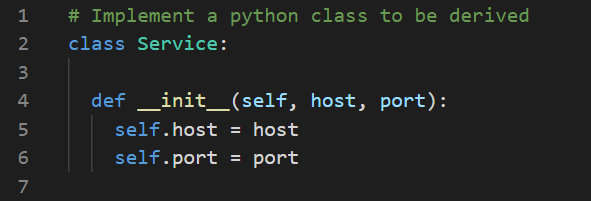
1. Then the final step is to save the resulting data to a file in JSON format. We have a try block so we can catch any exceptions and writing the file. If the file already exists, and we don't have permissions to write it, or if we're trying to write a file into a directory we don't have permissions for. Then it's possible that an exception will be thrown. If there's an error in converting the result into a JSON format, there can also be an exception. We handle those two cases.
2. On line 42, do an except of the IOError. And on line 44 do a blank except with no types, so it will catch all other exceptions. And in this case,just print the sys.exc\_info() the exception info. Go back to Line 40 inside the try block. So we do with open output file name as type w. So it's for writing as f, and then I call json.dump. We give it the object we're going to dump, we'll convert it into a JSON string. The file we're writing it to and the indent for the formatting. So it's going to indent the results by 2, so that we can have a nicely formatted JSON in this file.

[](https://cdn.filestackcontent.com/Xo5UZwlTQUh4bFCJ0OJQ)

## Solution: Object Orientated Programming

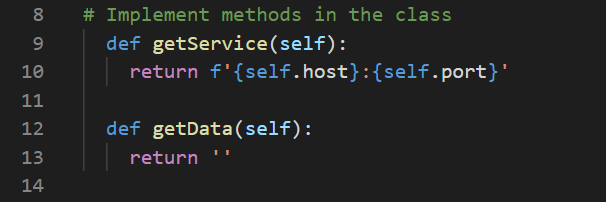
## Task 1

1. The first step is to implement a Python class to be derived. So the class we've created is called Service. This will simulate a base class for a service running on a particular host and port served over the Internet. We have class Service. And then we defined a \_\_init\_\_ method to be called when the class is created. We have self, host, and port as its parameters. And I set self.host = host and self.port = port.

[](https://cdn.filestackcontent.com/6G7R1PNiTFQVsKKSQ6LQ)

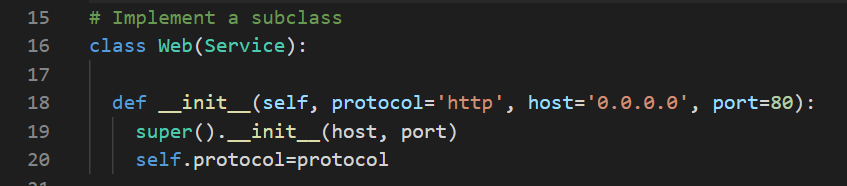
## Task 2

1. The next step is to implement methods in the class. We implement two methods, getService and getData. On line 9, def getService with self, so it's a member method. Then return f string, or formatted string, with {self.host} : {self.port}. And then getData returns an empty string because at this point the service is not defined. It needs to be subclassed for a particular implementation.

[](https://cdn.filestackcontent.com/FJOsl3ySXeS6wjhCq6ZV)

## Task 3

1. The next step is actually to implement a subclass. Scroll down, and on line 16 I have class Web. And then in parentheses I have Service, the base class. Our subclass is Web, and it implements its own init method, so def\_\_init\_\_ with self. And it has a protocol, in this case http.
2. Now, you can imagine a Web service with an HTTPS protocol for a secure service. We have the protocol in there to simulate that, and then we have host = 0.0.0.0. We have a default host which sets to the 0 host and port=80, the default HTTP port. And in this method, we actually call the base class init method using super(). We get the super, so this is the base class implementation, \_\_init\_\_, and we pass host and port to it. So we don't need to repeat ourselves and carry out the initialization in service. We don't need to repeat it here in Web. And then I set self.protocol=protocol.

[](https://cdn.filestackcontent.com/kjtMdNMQQUGN24Jmt6ii)

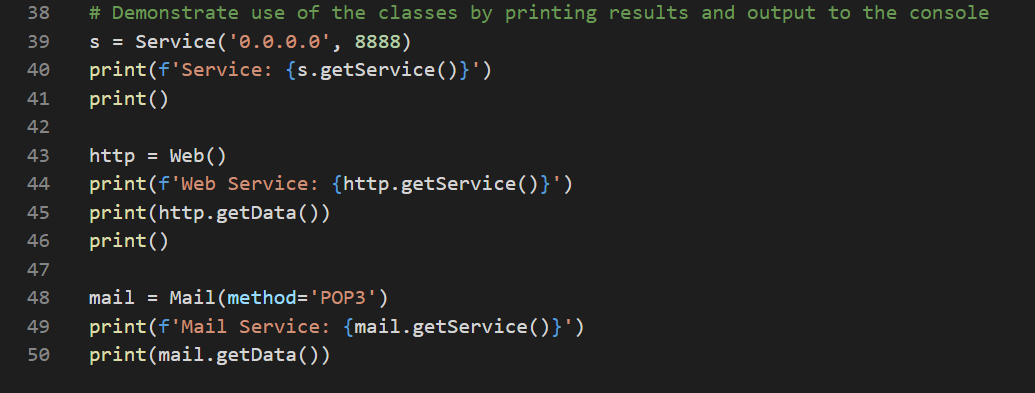
## Task 4

1. Then the next step of the exercise is to override methods in the subclass. We override getService and getData. GetService returns a formatted string with self.protocol://. We'd have the typical http:// followed by host:port. And then the getData, if it's a Web service we expect some sort of Web data to be returned, which could be some HTML. So just return html in the open and closed tags followed by ..., an ellipsis to simulate something being there. And then we close the HTML tag.
2. Now, for the sake of this exercise, to highlight the subclass, we implement a second subclass. Which wasn't required by the exercise, but we want to to highlight the subclassing in another way.
3. So on line 29, we have class Mail which subclasses from Service. It has its \_\_init\_\_ method, so def \_\_init\_\_ with self. It sets a method, so a mail method, and it defaults to IMAP. The host is 0.0.0.0, and the port is 143, the default port init, which is the default IMAP port. We call the super()\_\_init\_\_ method inside of this init with the host and port. And then we set self.method=method. And in this case, it overrides the getData method, and it returns something that simulates a mail message. It has a To, a From, and a Content-Type. And then a \n\n, so two new lines, nHello, World to simulate the message body. We'll see this in the output when we run the program.

[](https://cdn.filestackcontent.com/YmimpFCTduUlelwccZUA)

## Task 5

1. The last step of the exercise is to demonstrate use of the classes by printing results and outputting to the console. Create a Service, create a Web instance, and a Mail instance. On line 39, we have s = Service with 0.0.0.0 as the host and 8888 as the port. And print the Service call to getService. And print an empty print to put some space between this example and the next one.
2. And then on line 43, we have http = Web(). And then print Web Service with a call to http.getService, and print the getData method. So http.getData, and then an empty print to put a new line character in and put some space between this example and the next one.
3. Then we have the mail variable = the Mail class, where we specify a method = POP3, we specify a different mail method. On line 49, print Mail Service, the formatted string, with a call to getService. And then print mail.getData.

[](https://cdn.filestackcontent.com/zD6ITvMuTVOBUqABhE52)

## Terminal

1. Run the program and have a look at the Debug Console to see the results. We see the Service call to getService returns 0.0.0.0:8888, as expected. The Web Service returns the http:// with the host:port, the 0.0.0.0:80. And then the getData call returns the HTML tag example. And then the Mail Service returns something similar. It returns the host:port number and returns the contents of the mail message, translating the \n characters to actual new line characters. So we have To, From, Content-Type, and Hello, World as the message body. And that concludes this exercise.

[Text

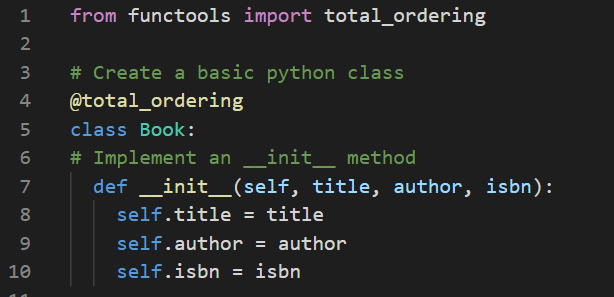
Description automatically generated](https://cdn.filestackcontent.com/PiACniVSOqHvebrX3tsw)

P5/17

## Solution: Special Method Names

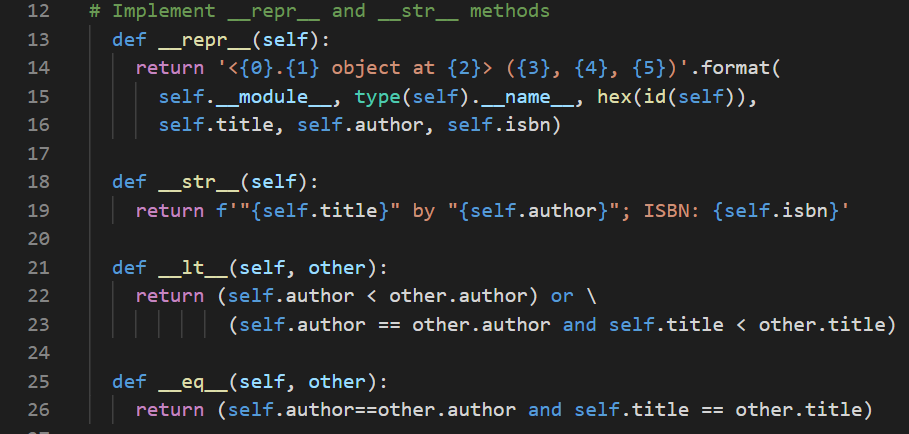
## Task 1 + 2

1. Start on line 1, from functools import total\_ordering. Which helps us compare items or compare classes just by implementing a couple of the ordering methods. And then Python will take care of implementing the rest.
2. The first step is to create a basic Python class. We have @total\_ordering, which is the decorator to direct this class to fill in the ordering methods based on what we implement. Implement a less than and an equals operator, and total\_ordering will fill in the rest. So it will fill in the greater than and less than or equal or greater than or equal, everything it needs to compare our Book class.
3. On line 5, define a class called Book. And then the next step is to implement a double\_init method. On line 7, I have def \_\_init\_\_ which takes self and then title, author and isbn. It's going to construct a record of a book with the title, author and isbn. Set self.title = title, self.author = author and self.isbn = isbn.

[](https://cdn.filestackcontent.com/O4vOu0DkR9GDgmOUXq2o)

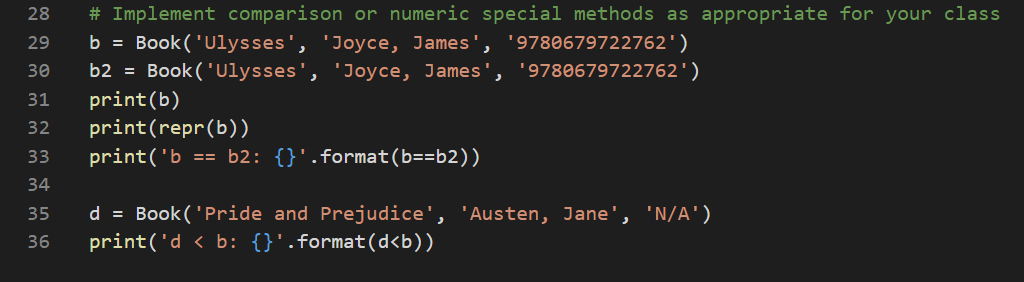
## Task 3

1. We set the member attributes or member variables to the parameters that are defined in this method. Then we implement the representation and string methods. Our \_\_repr\_\_ and \_\_str\_\_ methods. So for def \_\_repr\_\_, which takes a single argument, self, and it returns a representation of this class. It returns this formatted string with the braces and the variables enumerated. It has an open angle bracket and a closed angle bracket. It has the \_\_module\_\_, the type of the object or the type of the class, in this case book. Object at and then it has a hex id of self. It takes the id of self and it uses the hex value of that to say where this object is defined. It's almost like a memory location or a memory reference. And then it has in parentheses, the title, the author and the isbn. The representation is more for debugging purposes and the string representation is a human readable string.
2. We define them slightly differently in most cases. On line 18, define the \_\_str\_\_ method and return the f string of the self.title in double quotes by self.author in double quotes. `;isbn:v and then the isbn number for the book. And we need to implement comparison or numeric special methods as appropriate for our class.
3. Implement the less than equals and all the other comparison operators get populated. Because we've supplied the total\_ordering decorator above the class. For less than, we have def \_\_lt\_\_ with self and other. And our return, self.author, is less than other.author. We compare in alphabetical or lexicographical order the author strings. And if one is less than the other, then the book is less than the other book in terms of ordering. Then put a backslash so we can have a multiline statement. If self.author is equal to other.author, if it's the same author then we look at the title to see the order they should be in. So self.author equal to other.author and self.title is less than other.title then the books are less, then the books can be compared this way. So that's how we define the less than.
4. To define equality, so we have a def \_\_eq\_\_ with self and other as the parameters. In this case true gets returned if the authors are equal and the titles are equal between self and other. We'll implement these classes.

[](https://cdn.filestackcontent.com/vephf0U3QkycxeIQ7hnD)

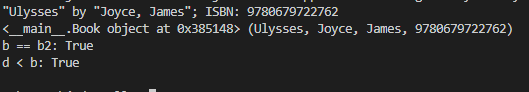
## Task 4

1. Create some books with authors and titles and isbns and we'll run some of the comparisons to verify the results. We have b = Book with Ulysses, by Joyce, James. So James Joyce, we put last name comma first name and then an ISBN number. I create a second book, b2 equal to the same definition with Ulysses, Joyce, James and the same isbn. So if we compare these two books they should be equal.
2. But first on line 31 call print of b. If you just supply the variable without anything else, it will take the string representation of the book. If I call the repr function on b, it will give us the representation of the book. And if I print b == b2, and check the result. It will print true or false whether or not these books are the same or have the same author and title.
3. And then finally on line 35, create another book called d = Book of Pride and Prejudice by Austen, Jane and fill in N/A for the isbn. And then on line 36, I print whether or not d is less than b. So comparing them which should be true because Austin in terms of alphabetical or lexicographical order, comes before Joyce. This should return true.

[](https://cdn.filestackcontent.com/wZUBzzPBTGS28juJsREL)

## Terminal

1. Now let's run the code and we can have a look at the results. So in this case, we get the string representation Ulysses by Joyce James with the ISBN number. Then we print the representation of it with main.book object at 0x and this hexadecimal value. And in parentheses Ulysses, Joyce, James, and the ISBN. And the way we've defined b and b2 with the same author and title, they are true and d is less than b. That returns true as expected because Austen is less than Joyce. And that concludes the exercise.

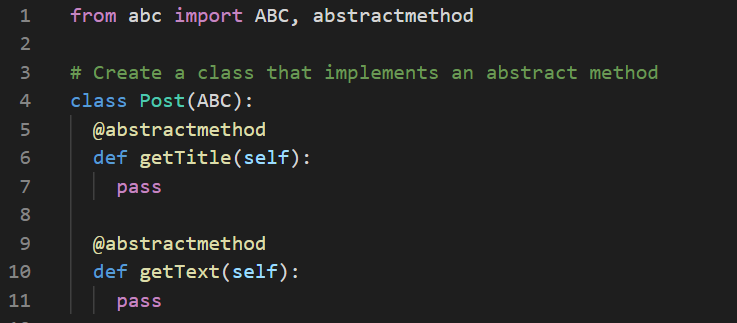
[](https://cdn.filestackcontent.com/Ys9lRvffRtSEn6E9kf1v)

7/17

## Solution: Static and Abstract Classes

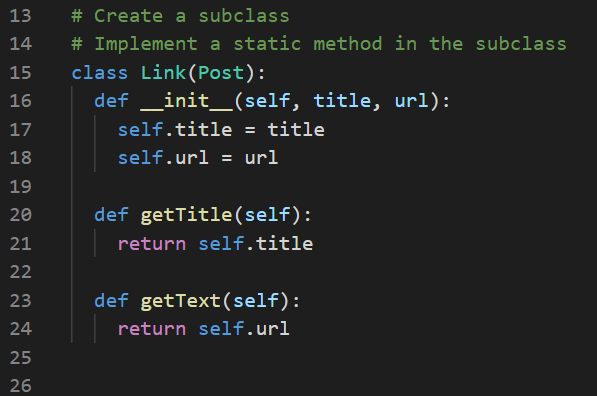
## Task 1

1. For this exercise start out by importing some important features or methods that we need to implement our abstractions. On line 1 we have from abc import the uppercase ABC, abstractmethod. The ABC stands for abstract base class and abstractmethod is a decorator for abstract methods.
2. The second step of the exercise is to create a class that implements an abstract method. Create a class Post(ABC) with ABC as its parent class. On line 5 we have @abstractmethod, we have the decorator. Then def getTitle with self to make it a member method or member function, and I define it with pass. It has the pass keyword because it doesn't implement a body to the method.
3. On line 9, there's another @abstractmethod decorator with another method def getText. We have two abstract methods in this abstract class called Post.

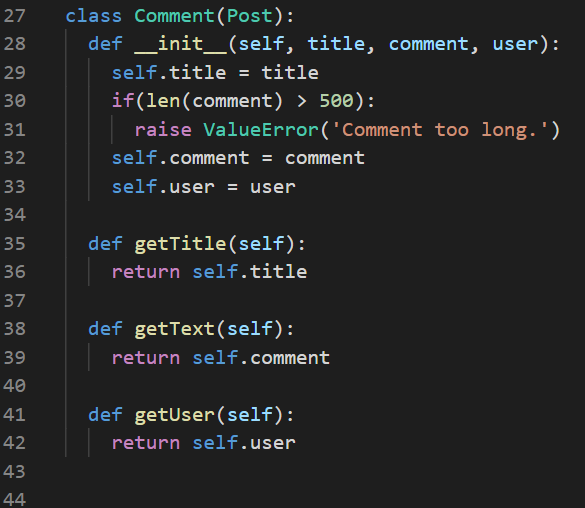
[](https://cdn.filestackcontent.com/95QwrypWQi62YmFHIoVs)

## Task 2 + 3

1. Now we need to create a subclass and implement a static method in the subclass. Those are the next two steps of the exercise. Create a class link which subclasses or inherits from Post, our abstract base class. It defines its \_\_init\_\_ method that take two parameters, a title and a URL, and then it sets the self.title and the self.url to their given variables.
2. On line 20, define the getTitle which is required because it's an abstract method in the base class. And it just returns self.title and then getText which returns self.url. The text of the link is the actual URL of that link. And we implement two more subclasses.

[](https://cdn.filestackcontent.com/FJLm1z2BRJqj4oIDqHy3)

1. The first one is comment. A comment is a type of post. We have class Comment(Post). In its \_\_init\_\_ method, it takes the parameters, title, comment, and user. Each comment has a title of the comment, the comment itself, the comment body or text, and the user that submitted the comment. On line 29, set self.title is equal to title. Check the length of the comment and if it's greater than 500 characters, raise a value error saying that the comment is too long. Otherwise I set comment equal to comment and self.user equals user. Implement the getTitle and getText methods, as well as a getUser method. And they return their respective variables. In the case of getText, the text of a comment is the actual comment body itself, so it returns self.comment.

[](https://cdn.filestackcontent.com/1V9LRFK4QFKc1MwxxMwL)

1. And then my third subclass of our abstract base class Post is an article. We have class Article(Post). In its \_\_init\_\_ method, it has a title and a body, so the body of the article, and it sets the respective variables of self.title and self.body. And it returns the title for getTitle and the getText method returns self.body.

[Text

Description automatically generated](https://cdn.filestackcontent.com/hQssCS0hQ7CPE0xIW5dc)

## Task 4

1. Now we need to instantiate the class and output its contents. Start by creating a list called posts. This will have the different posts, whether it be a link, a comment, or an article, and we'll add them to this list.
2. On line 59 create a link a = Link, that's called link title with a link http://example.com/link.html. Then call post.append(a), so we append this to the posts list. And then I print a.getText() and print an empty print statement to put a space between this and the next part of the output.
3. Then on line 64 we have b = Comment with the title as Comment title, the body of the text comment, and the user Steve. Create an article c = Article with Article title and Article text as its arguments. Then call posts.append(b) and then posts.append(c). Now loop over the posts and call getTitle on each of these posts, because they all implement the getTitle. On line 69, we have for p in posts, print p.getTitle.

[](https://cdn.filestackcontent.com/fDCMFwSRTFqnQN8wj4j1)

## Terminal

1. Run the code, And we get the output. We get the URL, so the getText called on the variable a, which was a link so it returns the link. And then in our for loop, the getTitle of each of the elements in the Posts list, are Link title, Comment title, and Article title as expected. And that concludes this exercise.

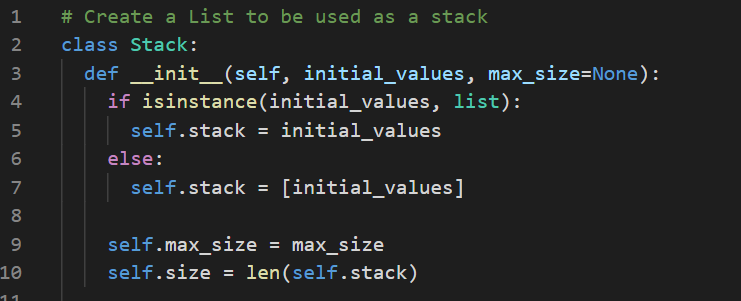
[Text

Description automatically generated](https://cdn.filestackcontent.com/1K6fDtcQLCnILFPPeDwf)

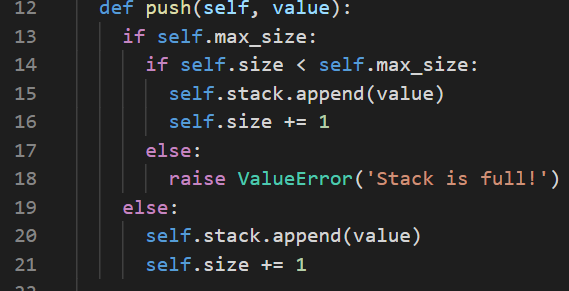
## Solution: Use a List as a Stack

## Task 1

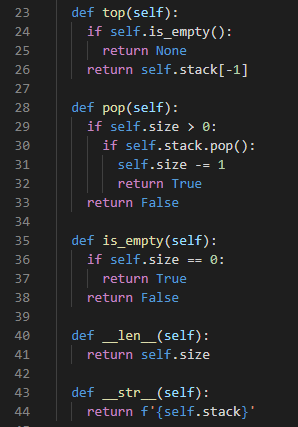
1. The first step in the exercise is to create a list to be used as a stack. Now you can do this directly, or you can incorporate the list inside of a stack class. We can implement some stack functions and carry out the exercise on the class itself. On line two define a class stack, and then I define the \_\_init\_\_ method to be used when the stack is instantiated. It has initial values as a parameter and max\_size which defaults to none. But if we want, we can set a max\_size of the stack, and prevent it from pushing new elements on to the stack if it is full, if it's reached the max\_size. And then, inside the ¬¬\_\_init\_\_ method, check to see if isinstance of initial values is a list. So if it's a list, then we can set self.stack equal to those initial\_values, otherwise, we set self.stack equal to initial\_values wrapped in square brackets. We make the value itself if it's a single value, we make it into a list.
2. Then on line 9 we set self.max\_size = max\_size, which defaults to none. And self.size, the current size is equal to len(self.stack),

[](https://cdn.filestackcontent.com/m9yAY84ARgqREON5n12h)

1. On line 12, define the push function. push takes a value and it pushes it onto the list or appends it to the list. Then I check if self.max\_size:, so if the variable is defined, check to see if self.size < self.max\_size. If there's room on the stack, then append it to the stack variable to the list. Call stack.append value, and increment the size by 1, so + = 1, else raise a value error that says the stack is full. And then the else that corresponds to the max\_size variable, if max\_size is not defined, call self.stack.append(value\_, and self.size + = 1 to increment it.

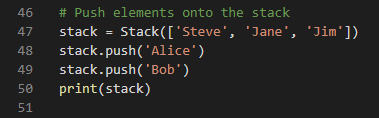
[](https://cdn.filestackcontent.com/zIvnDElRnuqjrTQiDV5V)

1. Define a top method, so many stacks have this top method that gets the top most element on the stack, but it doesn't actually pop it off. The pop checks to see if the stack is empty, so I call self.is\_empty. And if that's true, it returns none, otherwise it returns self.stack index that- 1. That's the last element in the list, which we defined to be the top of the stack.
2. On line 28, define the pop function, which checks to see if the size is greater than zero, and if it is, it calls self.stack.pop. Put the self.stack.pop in an if function because the pop function will return a value if there's one there. And then we can decrement the size by one and return true otherwise I return false.
3. My is\_empty method just checks to see if self.size is equal to zero and if it is it returns true. Otherwise it returns false, I implement the special function double underscore len. If the len function is called on our stack, it returns the size. And if a print function is called on our stack, or if it's turned into a string, it returns the string representation of the stack list. So it makes an f string of self.stack.

[](https://cdn.filestackcontent.com/p0jh9fnnT9O9yyiVyRUw)

## Task 2

1. Now the next step of the exercise is to push some elements onto the stack. On line 47, create a stack equal to the stack class with the initial values, the list of strings, these names Steve, Jane and Jim. Then call stack.push on Alice. The string Alice, put another name and stack.push Bob, and then I print the contents of the stack

[](https://cdn.filestackcontent.com/JVWD5boS2atjK5bmaebP)

## Task 3 + 4

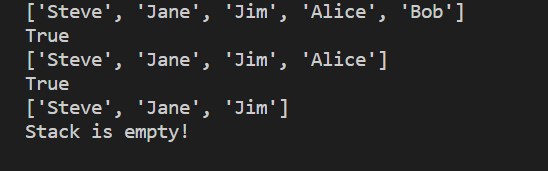
1. Now we need to pop elements off of the stack and output the contents of the stack. On line 54, call print stack.pop, which will return true if there's an element to pop. I print the state of the stack at that point after that pop, I call pop again, so stack.pop, and then print the stack.
2. One way we can iterate through all of the elements popping elements off of the stack is inside of a loop. On line 59, we have while len(stack) > 0, so while there are elements on the stack call stack.pop. And after this executes the len of stack at this point should be zero and it should print stack is empty.

[Text

Description automatically generated](https://cdn.filestackcontent.com/9lkmrg9FStu7GOEYpcn9)

## Terminal

1. Now run the code, and then in the debug console, we get the output of the stack at various points in the program. So we start with Steve, Jane, Jim, Alice and Bob, so we initialize it with the first three names. Then we push Alice and Bob onto the stack, and then we call stack.pop. So that returns true, which leaves us with Steve, Jane, Jim and Alice and Bob is popped off at the top of the stack. We call pop again, which returns true and then Steve, Jane and Jim are left on the stack, and Alice is gone. Then we remove all of the elements in our loop, we pop all of them off the stack. And then at the end the stack is empty as expected, and that concludes the exercise.

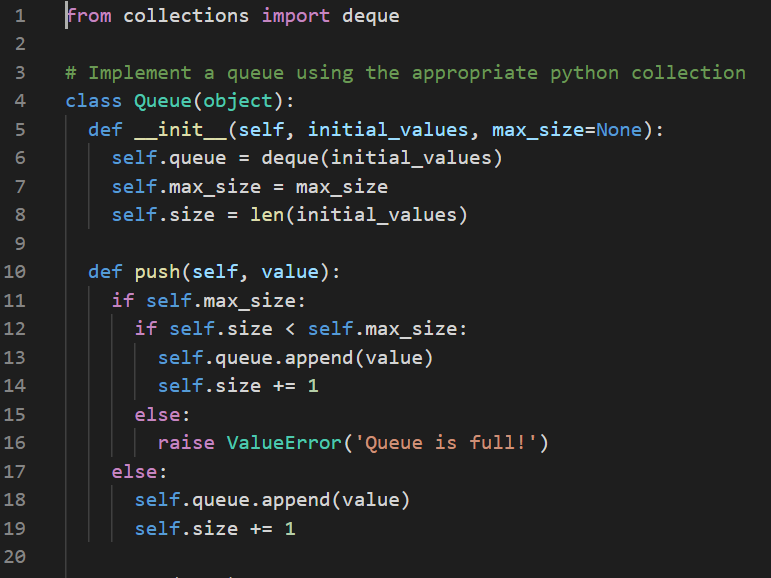
[](https://cdn.filestackcontent.com/iZiFyZSPayabCuh1ITOg)

11/17

## Solution: Python Queues

## Task 1

1. For this exercise, from collections import deque, which is short for a double ended queue, which is contained in the collections library from the Python standard library.
2. Now the first step of the exercise is to implement a queue using the appropriate Python collection. Well, the appropriate collection is a deque. Now, you could potentially do it with a list so you can implement a queue using a list, but it's inefficient to do so because of the way list is implemented. It's not recommended to use a list to implement a queue. If you're implementing a stack, a list is a perfectly fine data structure. But in this case with implementing a queue, we're going to use a deck. Now in our case, the queue is ‘a first in first out queue’, so a single ended queue, but we'll only be pulling off the front of the queue.
3. We have a class queue, which extends the basic object. And then define the \_\_init\_\_ function, which takes the initial values and the max\_size. We'll have an option to set a max size of our queue, and once it reaches that size, the queue will be full. Now this wasn't a strict requirement for this exercise, but include it so we can demonstrate how this could be done.
4. And then on line 6, I set self.queue = deque with the initial\_values. Set self.max\_size = max\_size. And self.size is the number of elements in the initial\_values or the len of initial\_values. On line 10, define a push function, so we can push elements into the queue. And this takes a value that we're going to push. And then inside the body of the function, check to see if self.max\_size: is set. If it's none, I just append the element to the queue, and increment the size by 1. So call self.queue.append parsing value as the argument, and set self.size += 1, to increment it by 1.
5. And back on line 12, if the max size is set, I make sure that the current size, self.size, is less than the max size. If it is, I call self.queue.append(value) and self.size +=1. Otherwise, raise a value error saying that the queue is full.

[](https://cdn.filestackcontent.com/GZIIvfqKQlOAaZHJSkJF)

1. The next method is a method called top. This is the first element or the current element in the queue that's next to come off. Checked to see if the queue is empty, so if self.is\_empty() is true then return none, so there's no element in the queue. Otherwise, return self.queue at index 0, so this is the first element in the queue. On line 26, define def pop, so it's a pop function that will pop the current element off of the queue. Check to see if the size is greater than 0. If it is, then there are items that we can pop from the queue. And then call self.queue.popleft. Popping left pops the first element off, and popping if we just call pop on queue, it will pop the rightmost element off. But that would be pulling from the queue from the wrong end because we want it in ‘first in first out’ fashion. So if this is true, if there's an element to pop off, then decrement the size. So self.size -=1, and return true saying that the pop was successful, otherwise return false. If we call pop on an empty queue, it will return false.
2. Then on line 33, we have def is\_empty, so this is an empty function, checks the self.size and if it's 0 it returns true, otherwise it returns false. Define a \_\_len\_\_ special function to return the size. If the len function is called on the queue, so I returned self.size, and the \_\_str\_\_ or string representation of the queue, which returns the f string of self.queue, which will output the values that are currently in the queue.
3. Now we can continue with the exercise, start by initializing the queue on line 44, so we have queue = Queue class. So we're creating an instance with the list of initial values being Steve, Jane and Joe, and the max\_size to be 6. We can have 6 elements in this queue before we reach the limit. Then call print len of queue to see how many elements are currently in the queue. At this point, it should be 3. Print an empty line.

[Text

Description automatically generated](https://cdn.filestackcontent.com/7glhBKZCTJW6gEhS24xk)

## Task 2

1. The next step of the exercise, is to add elements to the queue. So we call queue.push on the following strings, the following names, Alice, Bob, James and Sally. So if we count how many names we have to this point, we have three initially, and then we add four more. Sally will be the seventh name, so we should see a problem here with the max size if it's done correctly.
2. And then after that on line 53, print the queue, print the len of queue, and then print an empty line.

[Text

Description automatically generated](https://cdn.filestackcontent.com/zdcrKspwTxaDGQfDMvSC)

## Task 3

1. Now we are going to pop the elements. So we're popping elements from the queue in ‘first in, first out’ fashion. The first one in is Steve, so that's the one that should be popped out. On line 58, print queue.top to make sure it is Steve then call queue.pop, and print the queue. Push another name, queue.push Mallory. Print this state of the queue at that point, and then print queue.top, and an empty print.

[Text

Description automatically generated](https://cdn.filestackcontent.com/QwvIc0XhRA2D5dnXXufr)

## Task 4

1. Then I display the final contents of the queue by printing queue. And then on line 69, while len of queue is greater than 0, I call queue.pop to remove everything off of the queue.

[Text

Description automatically generated](https://cdn.filestackcontent.com/vNw0yhSHQR2QHPUWJQHB)

## Terminal

1. Now We run the program and see what happens in the debug console. At the very bottom, we see value error queue is full, and that's what we expected when we pushed Sally onto the queue. If we scroll up, we get the three printed from line 45 or we printed the len of queue, but then it only gets to line 52 before it fails. So we’re going to comment out queue.push of Sally on line 52, so that we can see the rest of the program execute successfully.
2. Now run the program again, and in this case, we get more output. We get the queue printed which is a deque from line 53 with Steve, Jane, Joe, Alice, Bob and James. And then call the length which is 6. And then top at this point is Steve, call pop, and then print the queue again. In this case, it starts at Jane, and then Joe, Alice, Bob and James with Steve being popped off the front of the queue. And then push Mallory onto the queue. And then at this point, the top of the queue is Jane, so when we print the results, we have Jane, Joe, Alice, Bob James, and Mallory. And that concludes this exercise.

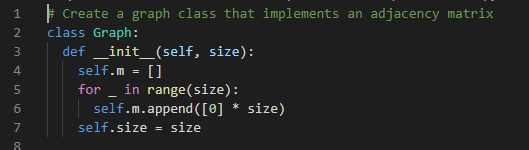
[Text

Description automatically generated](https://cdn.filestackcontent.com/zqt47u4pQ6kQguO6Eniu)

## Solution: Graphs as an Adjacency Matrix

## Task 1

1. The first step in the exercise is to create a graph class that implements an adjacency matrix. On line 2, we have a class graph. And in this graph, define a \_\_init\_\_ method that takes self and size as parameters. Inside the body of the class graph, define a \_\_init\_\_ method that defines a single parameter size. It's going to be a square matrix of the given size, its size by size. Then on line 4, define the matrix, self.m equals the double brackets. These square brackets define a list. It's going to be a list of lists to define our adjacency matrix.
2. And then for \_ in range(size), so we don't need the loop variable, so I just put it as an underscore and we loop size times. We're going to do self.m.append, and we're going to have 0 array times size. This is an adjacency matrix that's initialized to all zeros. Then self.size = size.

[](https://cdn.filestackcontent.com/LRznHbUgRIa4ci0NqUBy)

## Task 2

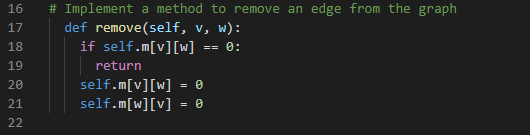
1. Now we need to implement a method to add an edge to the graph. It's to set the appropriate cell in our matrix to be 1 to designate an edge. The parameters that define the nodes are v and w. On line 11, We check to see that v = w. If it is, then we raise a ValueError saying that we cannot add an edge to itself. Then on line 13, we set, self.m of vw, so we're indexing at v and w equal to 1, and we need to set self.m[w][v] = 1. So an edge from one node to another and back so that edge is complete.

[Text

Description automatically generated](https://cdn.filestackcontent.com/zL2zdP46TvCynCmrLfH3)

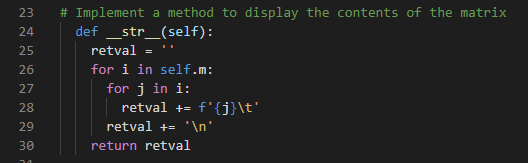
## Task 3

1. If there's an edge from node v to w, there's automatically an edge from w to v. Now we need to implement a method to remove an edge from the graph. We define a remove function that specifies v and w. Now if self.m[v][w] is already equal to 0, so if there's no edge, then we can just skip the removal. Otherwise, we set self.m at v and w equal to 0 and at w and v equal to 0.

[](https://cdn.filestackcontent.com/1WQV1wAjTEmXFIBtHKuQ)

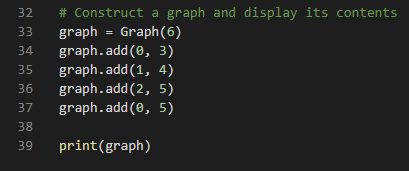
## Task 4

1. We need to implement a method to display the contents of the matrix. Overload the special string method the \_\_str\_\_ method, so that we can do a print on the graph instance and it will print the contents. On line 24, define this, and then in the body of this method, we have retval equals an empty string.
2. We start with an empty string, then for i in self.m:, so we loop over m, and we do it for j in i. So i will be a list, since m is the list of lists. In the inner list, we have for j in i:, and then we set retval plus equals the formatted string of the node value. Whether it's a 1 or 0, so that will be contained in j and then a tab character, so we separate each cell by a tab. And then retval += \n. So after each row, we start a new row on a new line, so that's why we need the newline character. And then in the end, we return retval, which should be our constructed string with the contents of the matrix.

[](https://cdn.filestackcontent.com/qd4JUlZQaS4egYetB33v)

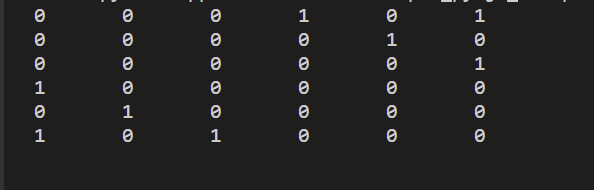
## Task 5

1. Then the next step is to construct a graph and display its contents. On line 33, set the graph, all in lowercase equal to the graph class with the uppercase G, and we set it to be 6. This is a size of 6, so it's a 6 by 6 graph, which means it has 6 nodes and the nodes are enumerated from 0 to 5. On line 34, we do graph.add(0, 3). So there's an edge from 0 to 3 and 3 to 0. And then graph.add(1, 4), graph.add(2, 5) and graph.add(0, 5). This constructs a graph with the edges as we defined here. And then on line 39, we just call print graph, which calls the string function we defined above.

[](https://cdn.filestackcontent.com/MHAArNrRLaxJxt4FFoxw)

## Terminal

1. Now we run the code, so we can see the results. And here we get our graph, so we have each row. So enumerated from 0, 0, 1, 2, 3, 4, 5. In this case if we go along the row from 0, 1, 2, 3, so this particular one is in the 0 row and the third column. If we're calling from 0 to 0 row and column 3, so that's our graph node from 0 to 3. And then the final one in the first row is our edge from 0 to 5.
2. Then in the next row, we have our edge from 1 to 4, and then our edge from 2 to 5 in the next row. Now in the bottom part, you'll notice that if you draw a line down the middle of the graph, down the diagonal, this graph is symmetrical. And it will always be symmetrical because it's an un-directed graph, which means if there's a node from 0 to 3, there's automatically a node from 3 to 0. And that concludes this exercise.

[](https://cdn.filestackcontent.com/5NHL7vRhQLCKaYcLppcY)

## Solution: Tree Traversal

## Task 1

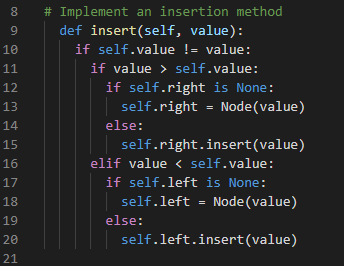
1. The first step is to create a class to implement a binary search tree. For this, we start with a class Node. And in this class, we define a \_\_init\_\_ method that takes a value, the value for that particular node.
2. On line 4, we have self.value = value. Since it's a binary search tree, we have two branches, which we typically call right and left. The right branch or a right node points to another node that has a value greater than the current one. And the left branch or left node will eventually point to another node that has a value less than the current value. The right takes greater values and the left takes lesser values.

[Text

Description automatically generated](https://cdn.filestackcontent.com/RbuRak8AQHaRII44sBrf)

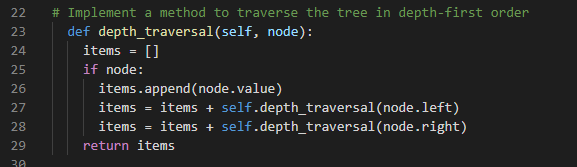
## Task 2

1. Then the next step is to implement an insertion method. On line 9, we have def insert, and this takes a value. What happens here is a bit of logic that's very important so that we can handle all of the cases of where to insert a node. The first thing we check if self.value, so if the current node is not equal to the value. Then we check to see if the value is greater than the current value, so greater than self.value. And if self.write is none, if it is a greater value, we look to the right branch. And if it's none, then we set self.write = node of the value.
2. We insert the node when we reach none. Otherwise, we call self.write.insert. We can think of it as recursively going down the rightmost nodes until we reach this place where we can insert the value. elif value is less than self.value. If the value is less than the current value of the current node, then if self.left is None, then we set self.left = Node(value). We create a new node, which left points to, otherwise we take self.left and we call its insert method with the value. Now what if the value is the same? Now in this solution, we’re assuming that the values are unique. If the value is equal, we assume that it's already there and we don't need to insert it again. So there are no duplicate values.

[](https://cdn.filestackcontent.com/d5d3CxDWQivLCgWbcUOg)

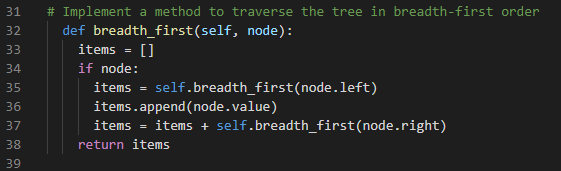
## Task 3

1. Then we need to implement a method to traverse the tree in depth first order. We want to go down the nodes of the tree till we reach the bottom, and then we insert the node value. On line 24, set this items variable equal to an empty list. We'll add the values to this list in the order that we traverse them. Now if node is set, so if it's none, we take items.append the node value. Then we take items = items + the result of self.depth traversal of the left nodes. And items = items + self.depth traversal of the right node.
2. We're going down the nodes, down the left nodes and down the right nodes. Now since this is being called recursively, we're calling depth traversal with each node starting with the left and going down the left most branches and then the right most branches. And this will recursively build up the items with all of the nodes that are traversed. And then in the end we return items, so it doesn't return items until it's traversed down the depth of an entire branch.

[](https://cdn.filestackcontent.com/WGXFzWeTni2iTG8BxqK1)

## Task 4

1. The next step is to implement a method to traverse the tree in breadth\_first order. There are a number of ways we can do this depending on how we view this breadth\_first. We want to think of going from left to right. We start at the left branch, and we move across the graph to the rightmost. So breadth\_first takes a node. It sets items equal to an empty list, so the items to be returned. And then if we have a valid node, we set items = self.breadth\_first(node.left). It goes to the leftmost, and we'll work our way across to the rightmost.
2. We take our left, then items.append(node.value), and then items = items + self.breadth\_first(node.right). We go from leftmost and then we queue up all of the rightmost. We're going across the graph. And we’re doing it this way so that we move across the graph, and this will pick up the nodes in sorted order. Now there are a couple of different ways we can do breadth\_first order. We can do it level by level going from left to right. Or we can also do it in a traversal that's picking up all of the nodes from the leftmost, which would be at the lowest level to the rightmost. And that's what I'm doing here. And then in the end, it returns items.

[](https://cdn.filestackcontent.com/vvhuynX5Tu2LNTrKWGUX)

## Task 5

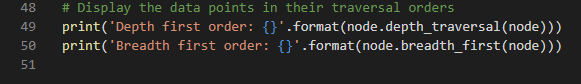
1. The next step is to create an instance of the tree and insert the data. Create a node = Node with value 10. We have a single node, we think of this as the root node, the first node we add. Then I called node.insert with the value 5, node.insert with 25, 12, 33, and 18. Insert these six values.

[Text

Description automatically generated](https://cdn.filestackcontent.com/9qJavudOT6KCRVXUzmBb)

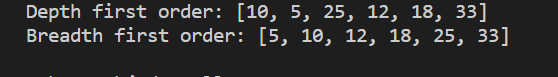
## Task 6

1. Then the final step is to display the data points in their traversal orders. print(depth first order: followed by the result of the node.depth\_traversal starting at the root node. And then breadth first order where we call node.breadth\_first and we insert that into the string.

[](https://cdn.filestackcontent.com/7QdPH8gxQHyv1ROGVRap)

## Terminal

1. Now if I run the program, We get the results printed in the debug console. So depth first order gives us 10, 5, 25, 12, 18, and 33. And breadth first order going from leftmost to rightmost, we get 5, 10, 12, 18, 25, and 33, giving us the results in sorted order. And that concludes this exercise.

[](https://cdn.filestackcontent.com/LfhGxIRoa2xk1pV7zeQM)