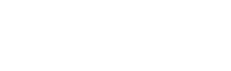
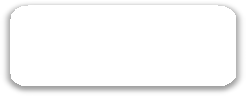
THEORY QUESTIONS ASSIGNMENT

Software Stream



**Maximum score: 100**

KEY NOTES

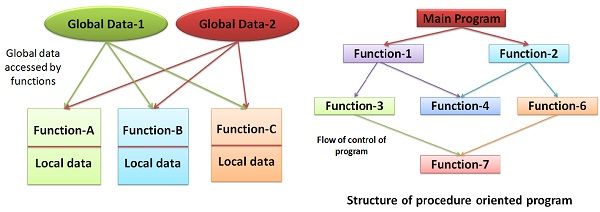
* This assignment to be completed at student’s own pace and submitted before given deadline.
* There are 10 questions in total and each question is marked on a scale 1 to 10. The maximum possible grade for this assignment is 100 points.
* Students are welcome to use any online or written resources to answer these questions.
* The answers need to be explained clearly and illustrated with relevant examples where necessary. Your examples can include code snippets, diagrams or any other evidence-based representation of your answer.

|  |  |
| --- | --- |
| **Theory questions** | **10 point each** |

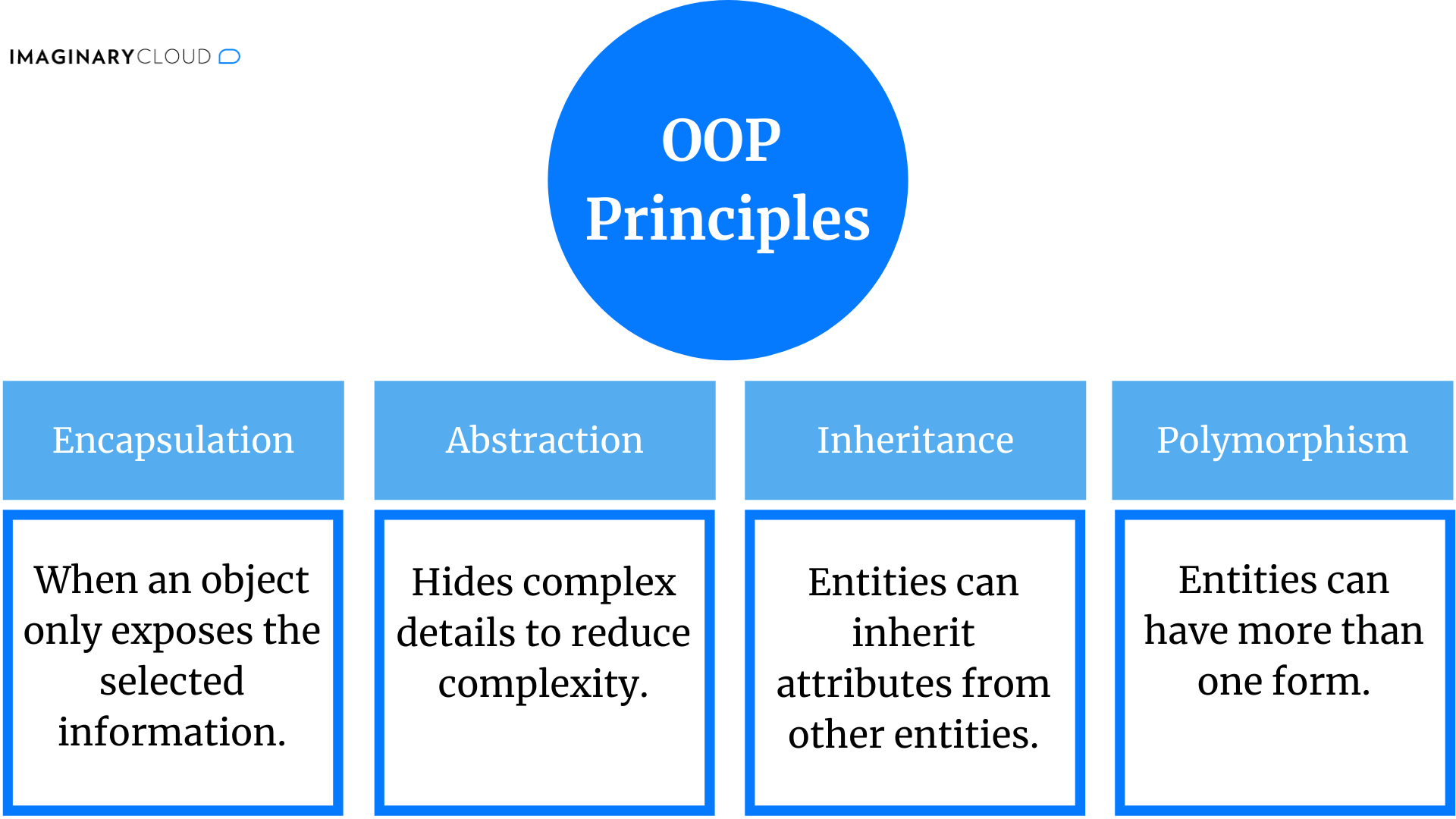
1. **How does Object Oriented Programming differ from Procedural Oriented Programming?**

Object Oriented Programming (OOP) and Procedural Oriented Programming (POP) are two examples of programming paradigms (approaches) to solving problems and developing software. All programming languages follow at least one programming paradigm (some use multiple paradigms) and each has its own strengths and weaknesses depending on the requirements of the application.

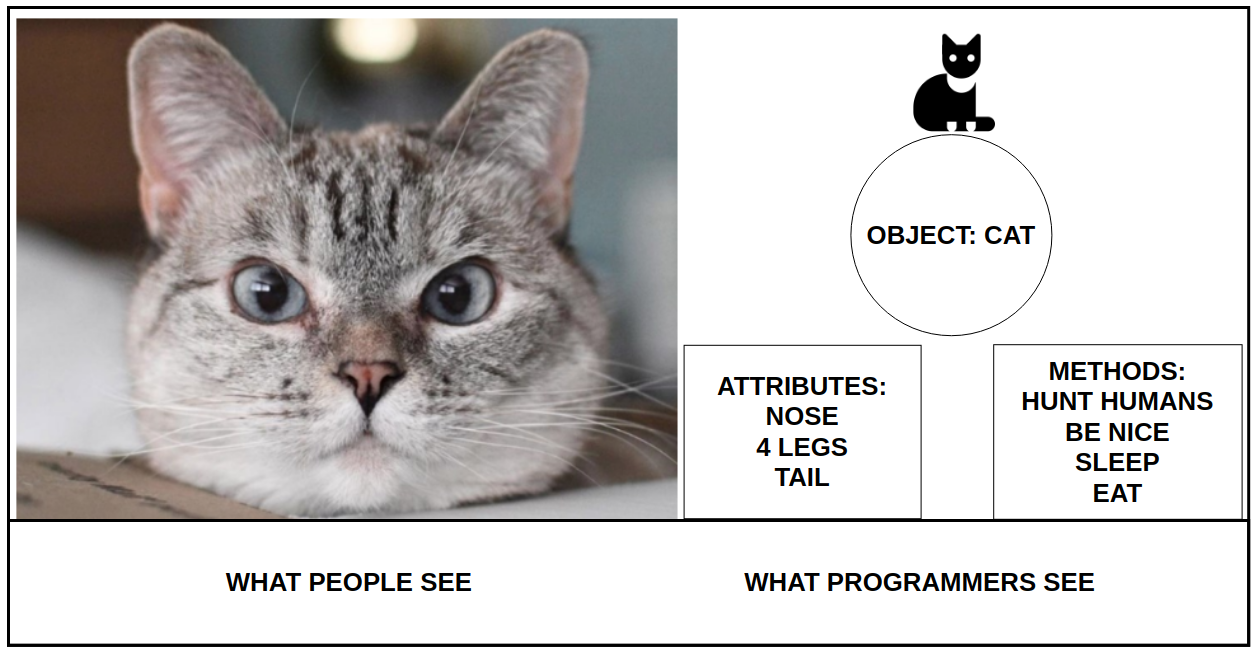
Procedural Oriented Programming (POP) is derived from structured programming and takes a top-down approach, completing tasks in sequential order. It is based on procedures (routines, subroutines, functions) – sets of step-by-step instructions and computations to be carried out. When a program is executed, any given procedure can be called at any point. Large programs are structured in small units that share global data. This is a data security concern as unintentional changes can be made in the program by functions. Languages used with PP – FORTRAN, ALGOL, COBOL, BASIC, Pascal and C.



Object Oriented Programming (OOP) is based on the concept of objects that relate to the real world. They contain data (attributes) and behaviours (methods), which interact with other elements in the application. For example, an object could be a cat. That cat would have a name (a property of the object) and would know how to ‘meow’ (a method). A method in OOP is similar to a procedure in PP, the difference being that the method belongs to a particular object. The most popular OOP languages are class-based – a class can be considered as a blueprint for an object; therefore, objects are said to be instances of classes. Pure object-oriented languages follow the four core principles: encapsulation, abstraction, inheritance, and polymorphism. Languages used with OOP – Java, C++, C#, Python, PHP, JavaScript, Ruby, Perl, Objective-C, Dart, Swift, Scala.

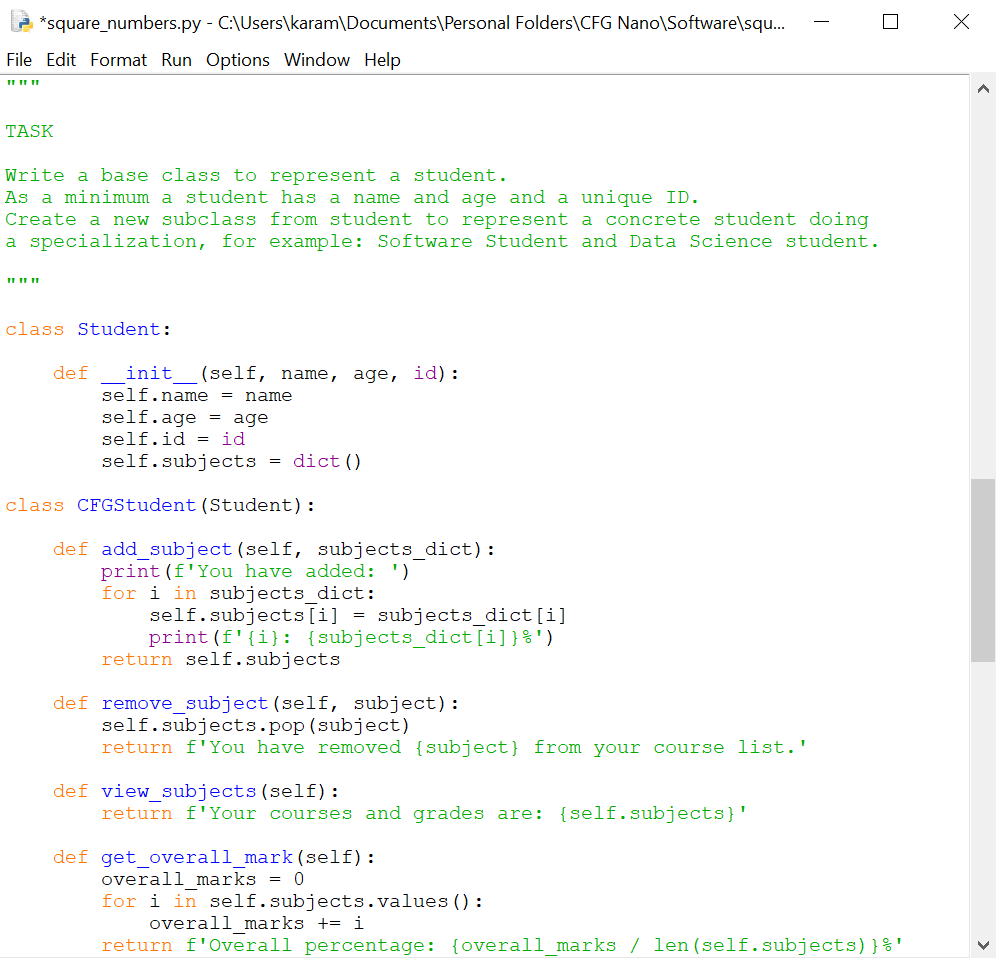


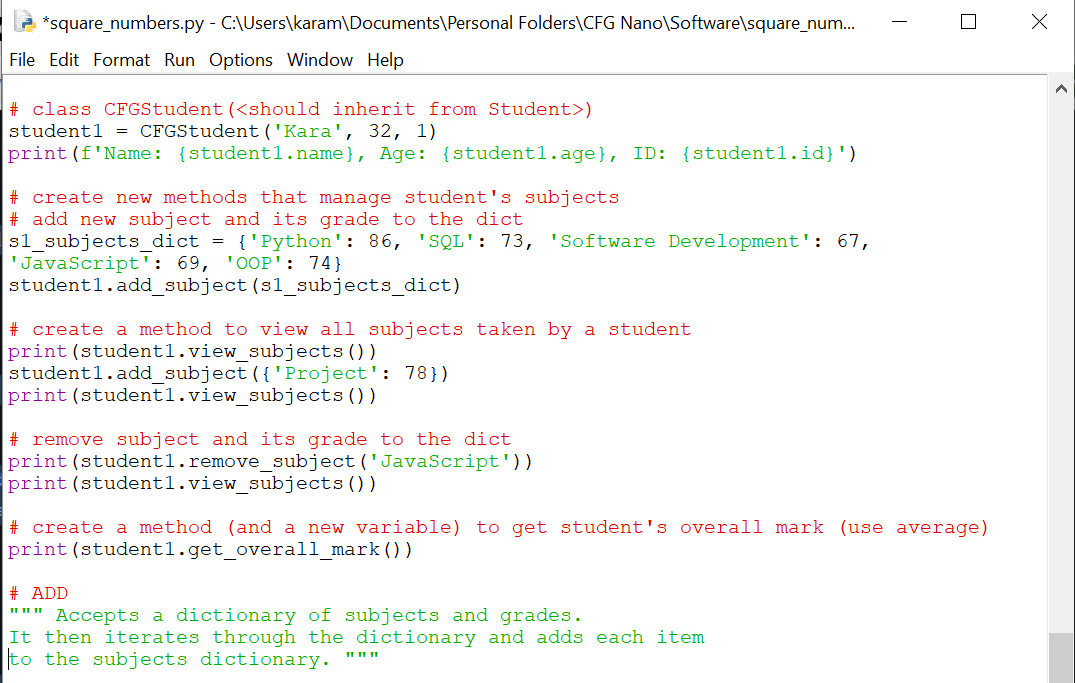
OOP can have several advantages over POP, especially if the task is highly complex. As data is grouped with the functions that operate on it, it is more secure. Access to certain attributes can also be restricted. Code is more reusable with OOP as once one class is created, multiple instances (objects) can be created. Lengthy POP programs are prone to flaws and can be difficult to read and understand if not organised carefully, and accessibility of data throughout the program can make it vulnerable.

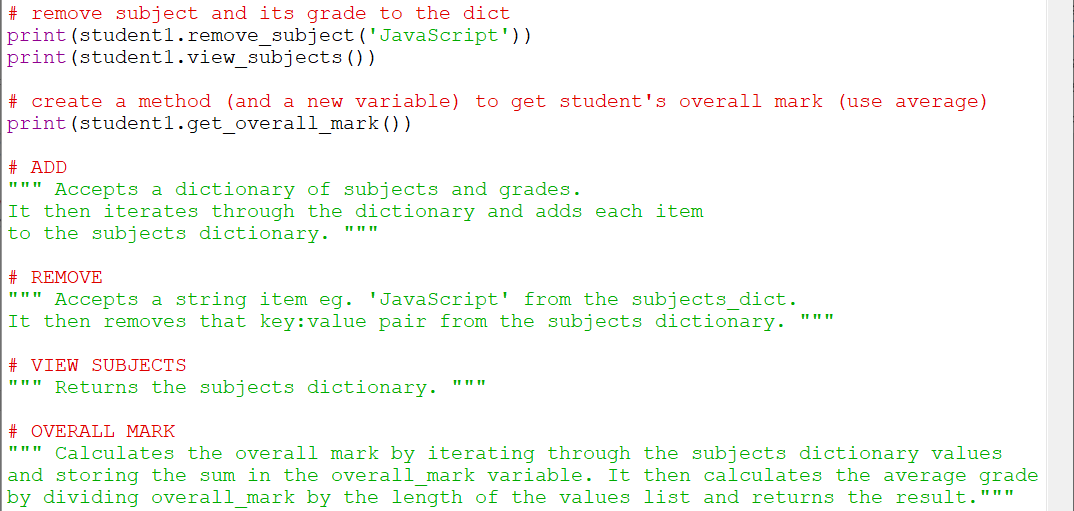


Example of OOP:

*Student class with name, age, ID, subject/grade (dict) attributes. CFGStudent subclass inherits attributes from Student class and has its own methods for adding, removing, and viewing subjects and for getting the overall mark (average grade). The Student class is like a template from which several different types of students could be created.*







| **BASIS FOR COMPARISON** | **POP** | **OOP** |
| --- | --- | --- |
| Basic | Procedure/Structure oriented. | Object oriented. |
| Approach | Top-down. | Bottom-up. |
| Basis | Main focus is on "how to get the task done" i.e., on the procedure or structure of a program. | Main focus is on 'data security'. Hence, only objects are permitted to access the entities of a class. |
| Division | Large program is divided into units called functions. | Entire program is divided into objects. |
| Entity accessing mode | No access specifier observed. | Access specifier are "public", "private", "protected". |
| Overloading or Polymorphism | Neither it overloads functions nor operators. | It overloads functions, constructors, and operators. |
| Inheritance | There is no provision of inheritance. | Inheritance achieved in three modes public private and protected. |
| Data hiding & security | There is no proper way of hiding the data, so data is insecure | Data is hidden in three modes public, private, and protected. hence data security increases. |
| Data sharing | Global data is shared among the functions in the program. | Data is shared among the objects through the member functions. |
| Friend functions or friend classes | No concept of friend function. | Classes / function can become a friend of another class with keyword "friend". (used only in c++) |
| Virtual classes or virtual function | No concept of virtual classes. | Concept of virtual function appear during inheritance. |
| Example | C, VB, FORTRAN, Pascal | C++, JAVA, VB.NET, C#.NET. |

**Advantages**

POP (Procedure Oriented Programming)

* Ability to reuse the same code in various places
* Facilitates tracking of program flow
* Ability to construct modules

OOP (Object Oriented Programming)

* Objects assist with task partitioning in the project
* Data hiding allows secure programs to be built
* Can potentially map the objects
* Categorisation of objects into various classes
* Object-oriented systems can be upgraded easily
* Inheritance eliminates redundant code
* Codes can be extended using reusability
* Greater modularity can be achieved
* Increased reliability due to data abstraction
* Flexible due to the dynamic binding concept
* Information hiding decouples the essential specification from its implementation

**Disadvantages**

POP (Procedure Oriented Programming)

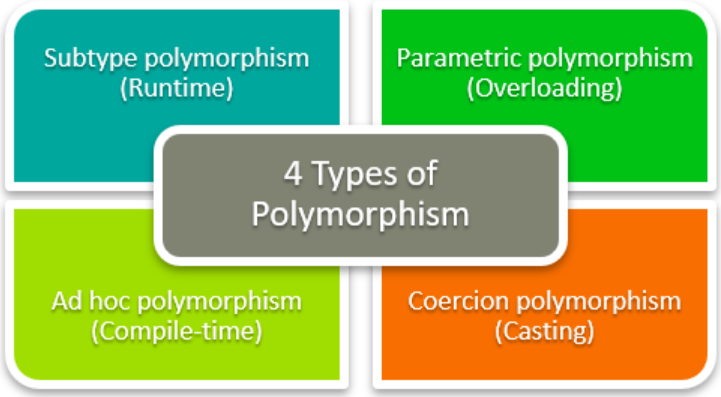
* Global data is vulnerable
* Data can move freely within a program
* Can be difficult to verify the data position
* Functions are action-oriented
* Functions are not capable of relating to the elements of the problem
* Real-world problems cannot be modelled
* Parts of code are interdependent
* Lack of reusability – one application code may not be able to be used in other application
* Data is transferred by using the functions

OOP (Object Oriented Programming)

* More resources are required
* Dynamic behaviour of objects requires RAM storage
* Detection and debugging can be challenging in complex applications when the message passing is performed
* Inheritance makes classes tightly coupled, which can affect the reusability of objects

1. **What's polymorphism in OOP?**

Polymorphism means taking many forms. In programming, it refers to the use of a single entity (method, operator or object) to represent different types in different scenarios. For example, the addition operator (+) has several uses. It is used for arithmetic addition with integers and concatenation with strings.

****

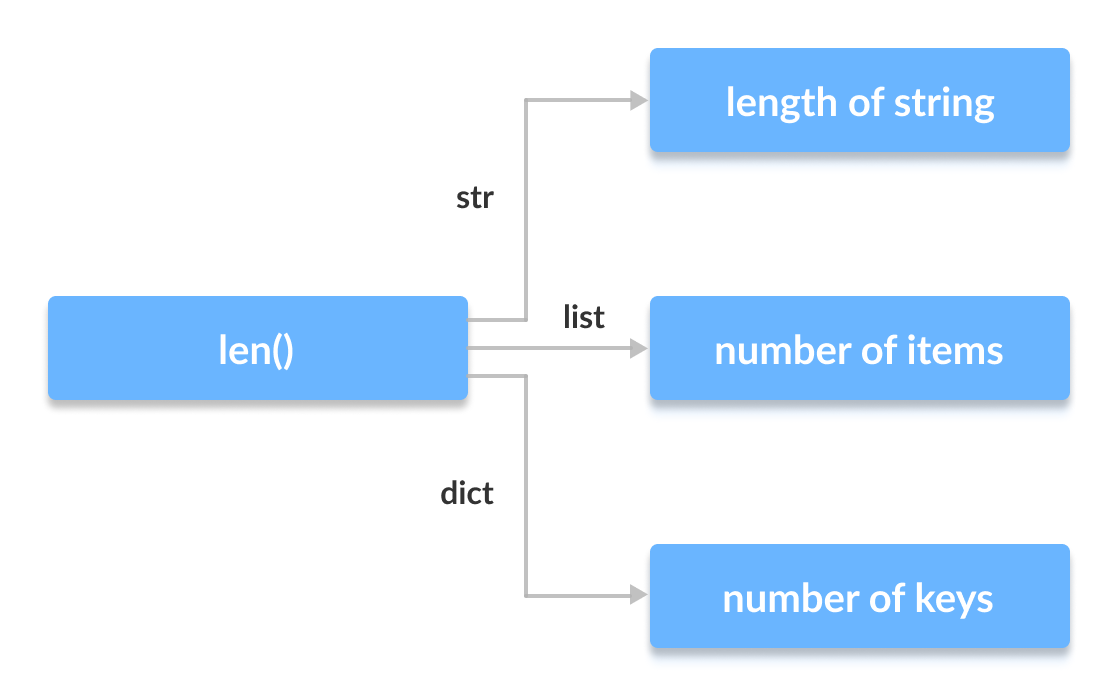
**Function polymorphism**

Some functions are compatible with various data types, such as len(). The len() function can be used with a string, list, tuple, set, or dictionary but return specific information about each.

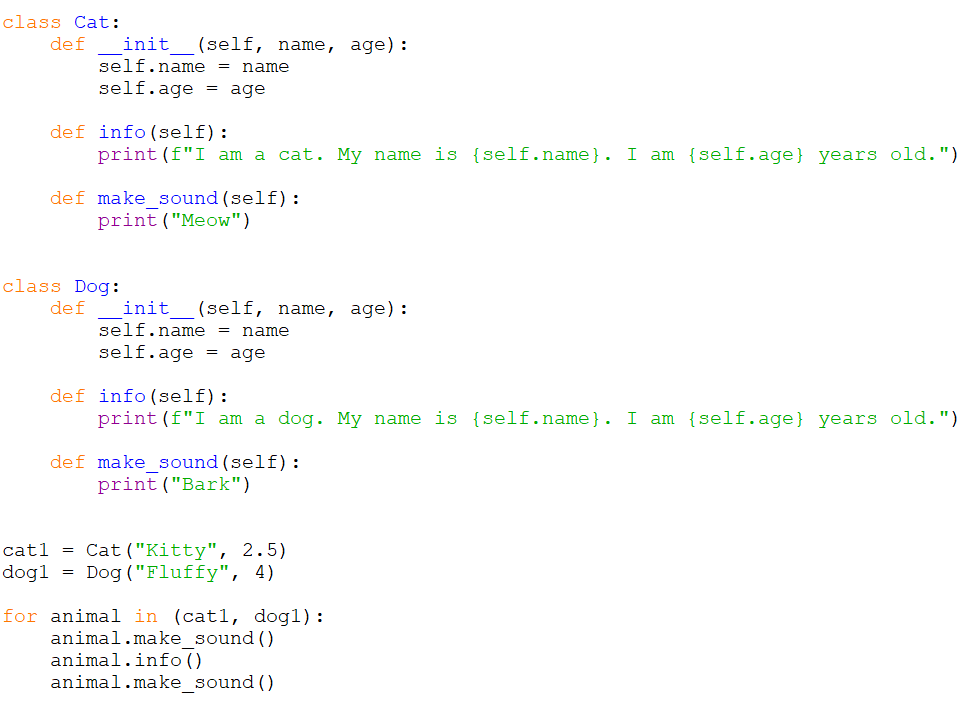
**print(len("Hello World!"))**

**print(len(["Python", "Java", "C"]))**

**print(len({"Name": "Kara", "Address": "London"}))**



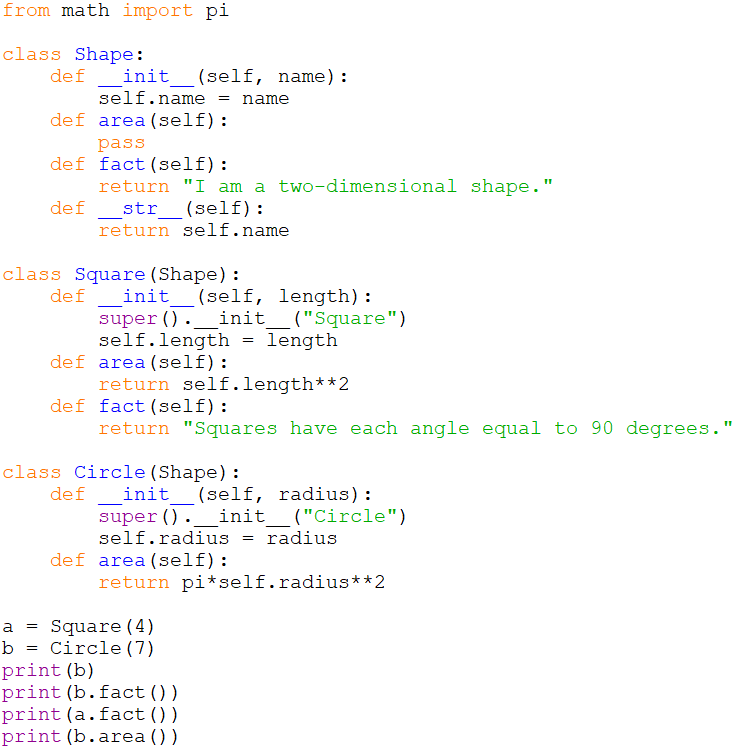
The concept of polymorphism can be used while creating class methods because Python allows different classes to have methods with the same name. These methods can be generalised by disregarding the object we are working with.



Two classes, Cat and Dog, have been created. Their structures are similar and both have the same method names info() and make\_sound(). There is no common superclass or link between the classes. However, due to polymorphism, these two different objects can be put into a tuple and iterated through using a common animal variable.

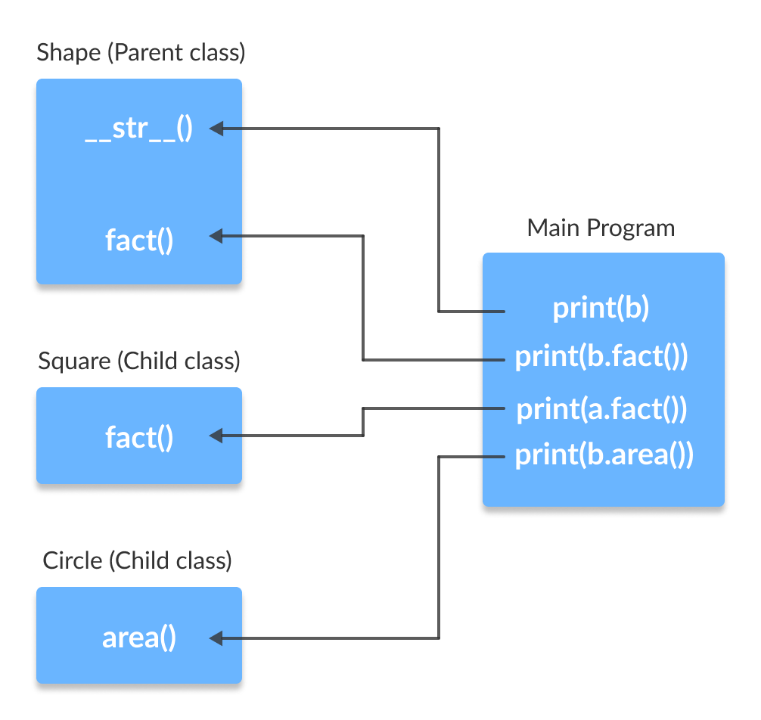
**Polymorphism and Inheritance**

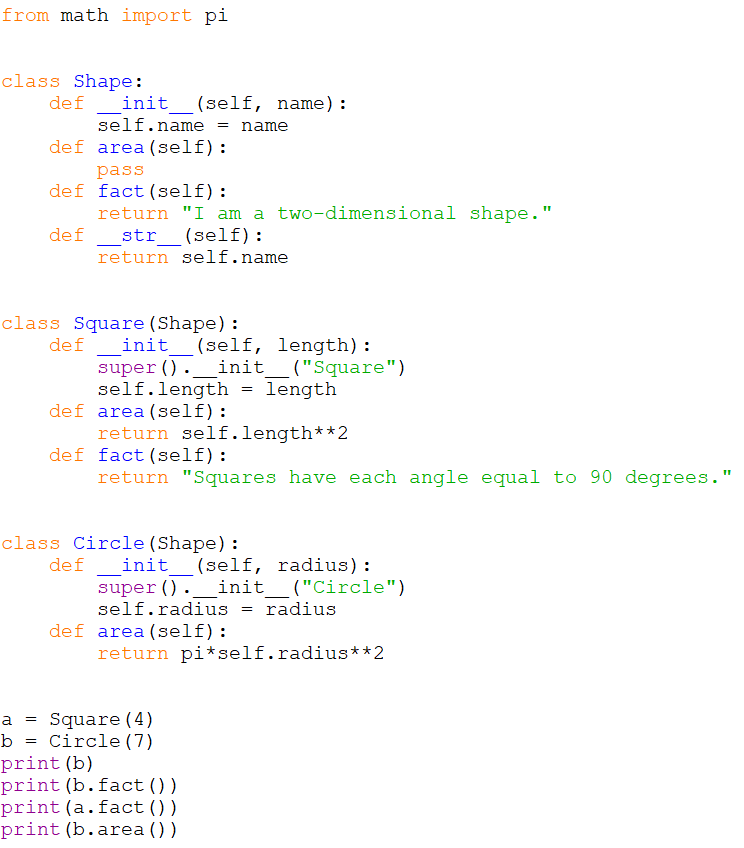
The child classes inherit methods and attributes from the parent class. Certain methods and attributes can be changed to fit the child class, which is known as Method Overriding. Polymorphism allows us to access these overridden methods and attributes that have the same name as the parent class.



**Method Overriding**

Methods such as \_\_str\_\_(), which have not been overridden in the child classes, are used from the parent class. Due to polymorphism, the Python interpreter automatically recognises that the fact() method for object a(Square class) is overridden and uses the one defined in the child class. The fact() method for object b isn't overridden, so it is used from the Parent Shape class.





1. **What's inheritance in OOP?**

Inheritance is a very useful concept in OOP. We use inheritance to inherit properties of one class to another class. It means we can use the existing features on one class in another class. Simply it provides the facility of code re-usability. We don’t need to write same code in application again and again.

Example of hierarchical inheritance:

*Person class (Super class/Base class)*

*Programmer, Dancer, Singer (Child classes)*

* Class that provides attributes/properties to another class is known as parent class. Parent class is also called a Base class or Super class.
* Class that receives attributes/properties from another class is known as child class. Child class is also called a Sub class or Derived class.
* Important: In inheritance, a parent class never obtains features from its child class.

Diagram

Description automatically generated

**Inheritance Types:**

Diagram, timeline

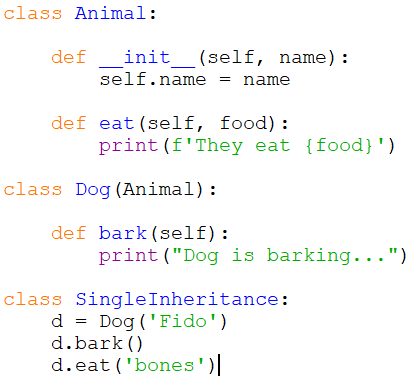
Description automatically generated

**Single Inheritance**

Diagram

Description automatically generated

The child class inherits properties and behaviour from single super class (parent class).



Super class is Animal and Sub class is Dog. We create a class called Animal and create a method eat(). If you add changes to Animal class it should affect whole code. We create a Dog class which extends from Animal class. Dog class contains all the properties of Animal class. So we can call features of both class.

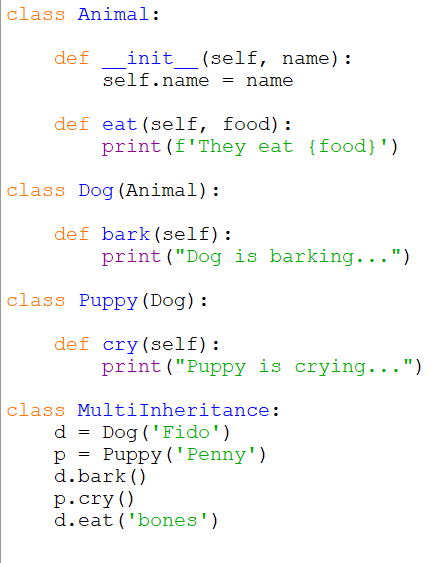
**Multilevel Inheritance**

A picture containing text, sign

Description automatically generated

Classes having more than one Super class at different levels are known as Multilevel Inheritance.

Here, Child1 class inherits the properties of the parent class. Child11 class inherits the properties of Child1 class. For Child11 class, Child1 class is a Super class. Child11 class implicitly inherits features of both Child1 class and Parent class.



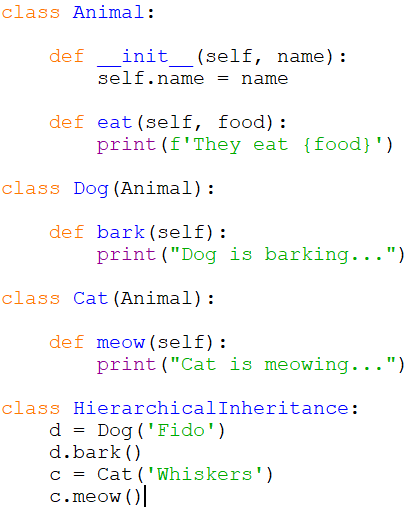
Puppy class can get all the properties in both Dog class and Animal class.

**Hierarchical Inheritance**

Diagram

Description automatically generated

If super class (parent class) has more than one sub class (child class) this is known as Hierarchical Inheritance. Both Child1 class and Child2 class are inheriting from Parent class.



Dog class and Cat class are Sub classes which are inheriting from Animal class (parent class).

**Hybrid Inheritance**

Diagram

Description automatically generated

Combination of Multilevel inheritance and Multiple inheritance.

1. **If you had to make a program that could vote for the top three funniest people in the office, how would you do that? How would you make it possible to vote on those people?**

• List of key requirements

* Frontend – access through website / app
* Search list of employees
* Voting buttons for 1st, 2nd, 3rd choices
* Visualisation of top 3 funniest people in the office
* Confirmation of voting
* Database – store employee and voter information
* Secure storage of data
* Maintain accurate information
* Backend – Python, API, server to get seat availability
* Secure access
* Fast and efficient retrieval of data
* Fault tolerant
* Locking of submitted selections

• Main considerations

* Scalability – how many users can use and access the voting system
* Accessibility – is the UI accessible to users with visual / cognitive / mobility needs
* Usability – user-friendly design, easy to access and navigate, quick and accurate
* Compatibility – system can be used on various devices e.g., PC / tablet / mobile
* Security – protecting user/subject data, passwords, backups
* Management / Maintenance – ease of use for administrators of the system

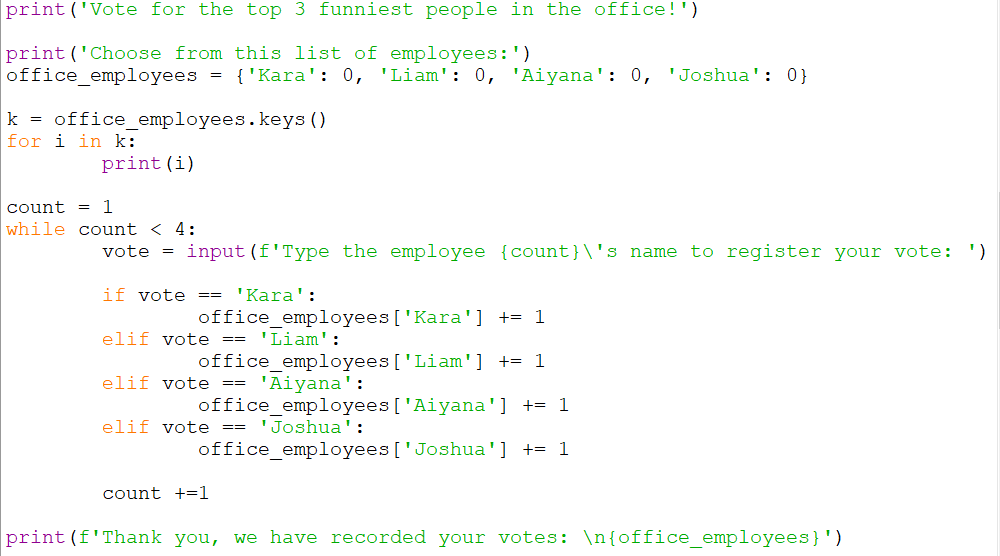
• What would be your common or biggest problems?

* Data protection / security / anonymity
* Consent from employees to feature on voting system
* Backup storage systems
* Manging traffic / scalability
* Maintaining up-to-date information e.g., consenting subjects
* Training administrators to use the system

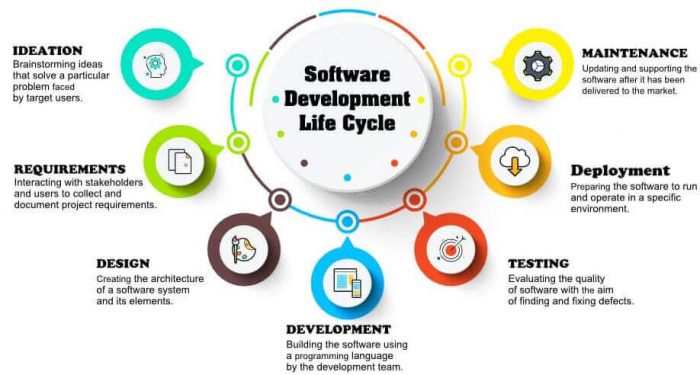
• What components or tools would you potentially use?

* Python celery for asynchronous code to make the user experience better (faster)
* DBMS – to manage locks for voting and to keep data accurate

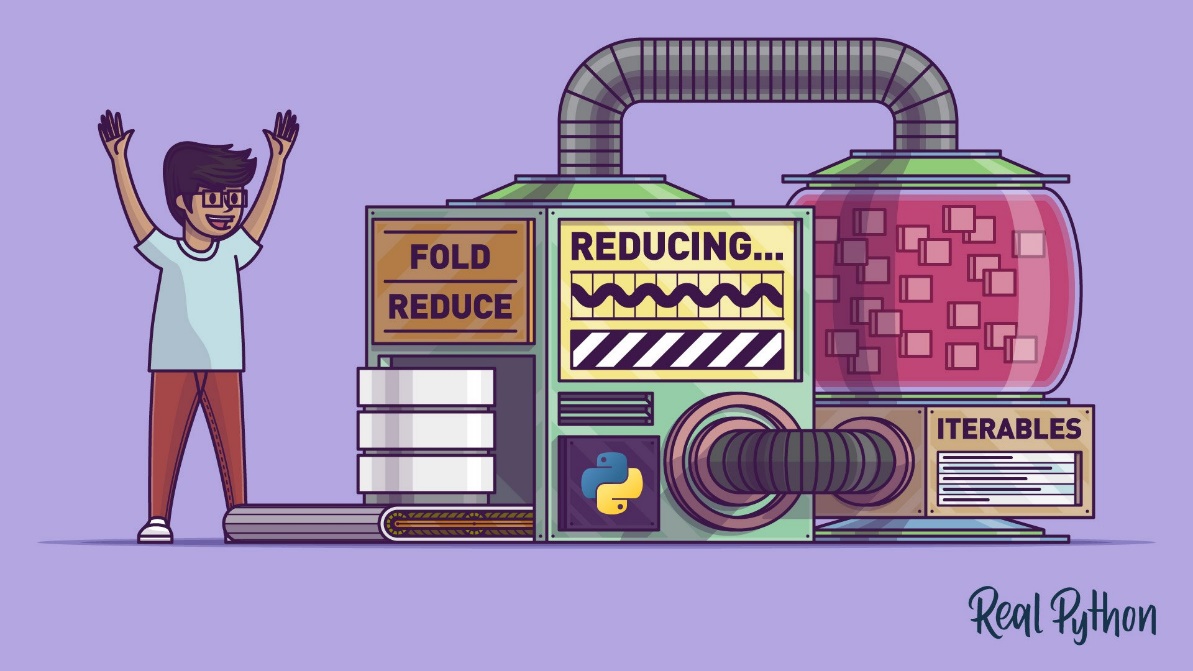
**Example:**



1. **What's the software development cycle?**



1. **What's the difference between agile and waterfall?**
2. **What is a reduce() function used for?**



The function reduce() implements a mathematical technique called folding or reduction to reduce an iterable to a single cumulative value.

1. **How does merge sort work**

Merge sort is a sorting algorithm that is made up of two subroutines – the split subroutine and the merge subroutine.

Base case – a single item or empty list is a sorted list

If base case, return item

n = 8 – we split until we reach a base case, then the merge subroutine is implemented.

n = 8 – 3 splits happen log(n)

n leaves in recursion call

2 to the power of 3 = 8

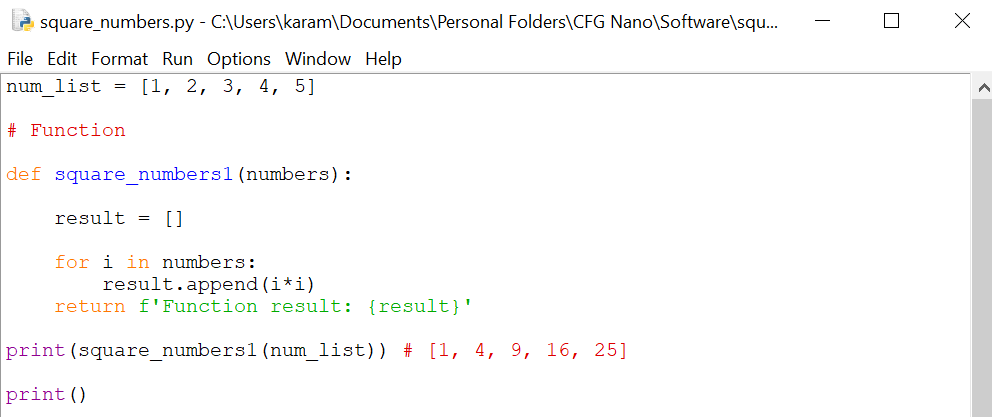
1. **Generators – Generator functions allow you to declare a function that behaves like an iterator, i.e., it can be used in a for loop. What is the use case?**

Functions can be transformed into generator functions with a simple change – instead of using the ‘return’ keyword, we instead use ‘yield.’ Generators return objects (iterators) by implementing \_\_iter\_\_() and \_\_next\_\_() methods automatically, but do not start execution until there is a call.

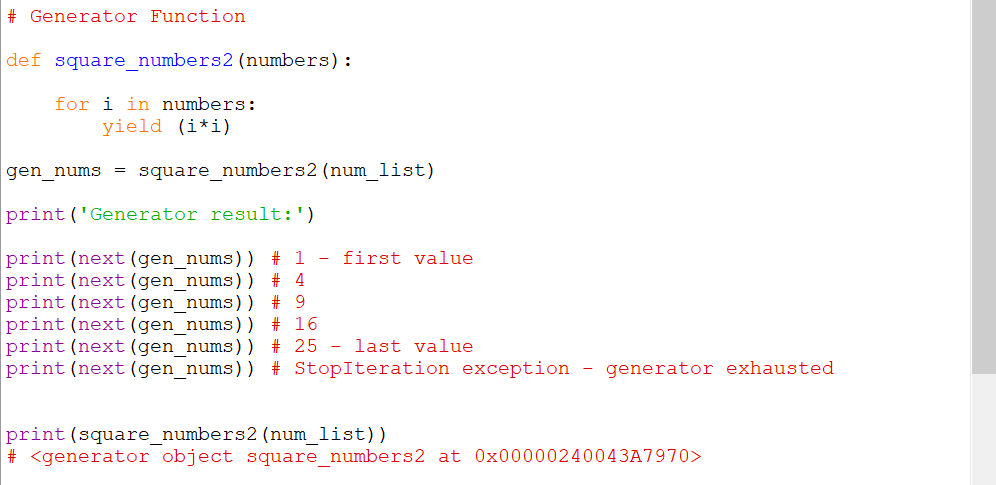
Generators don’t require as much space in memory as they yield one result at a time and keep state (track) of the current position in the object between calls. While ordinary functions terminate completely after ‘return’, the ‘yield’ keyword causes the generator function to pause on the current value, then the next() method causes the generator function to return the next value in the object. This efficient use of memory is particularly important for large lists and when memory space is limited.

Comparison of a function and a generator function:

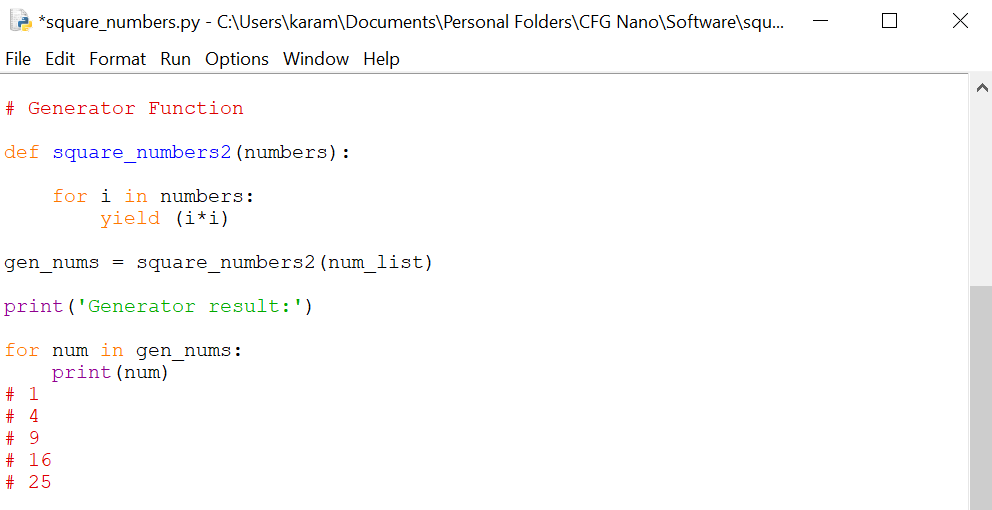
*The function square\_numbers1 takes a list of numbers and iterates through the list using a for loop. Each number is squared then the result is appended to the initially empty result list. This entire list is stored in memory and returned by the function.*

**

*The generator function square\_numbers2 does not require an empty list as it yields each value while keeping state. As 25 is the last value to be yielded, a Stop Iteration exception is thrown on the 6th time next() is used – the generator is exhausted and out of values.*



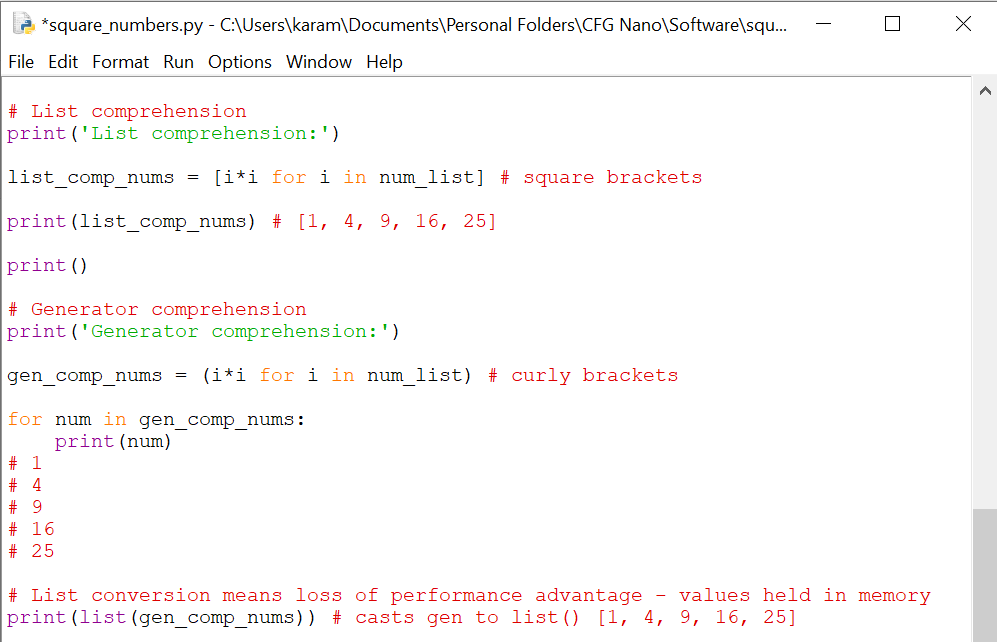
*For loops can also be used with generators to output each value in the object:*



List vs generator comprehension:

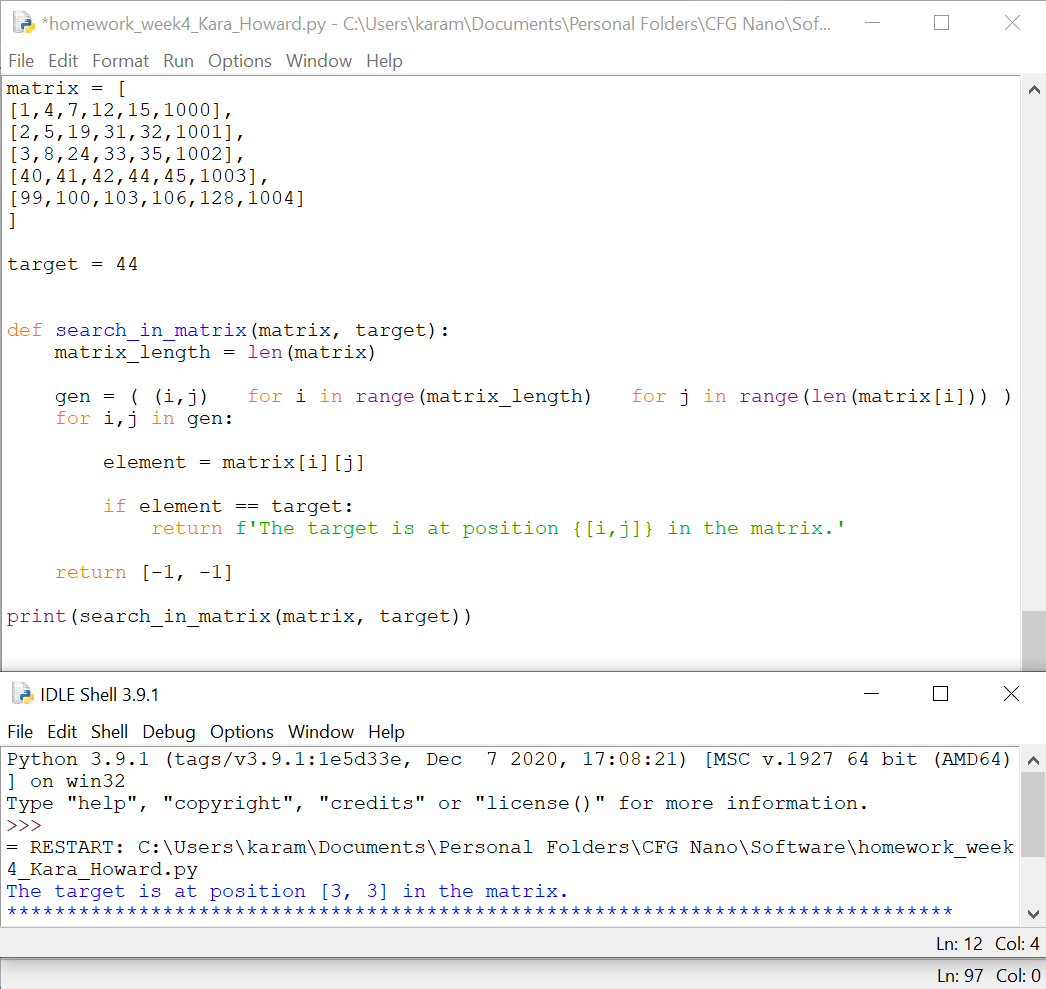
*Python generator expressions are similar to anonymous lambda functions. Generator comprehensions are often used as they are less memory intensive compared to list comprehension; however, they tend to take more time to run.*

*Instead of the square brackets used in list comprehension, generator comprehension uses curly brackets. They output one item at a time unlike list comprehension, which outputs the entire list. The output from the generator expression can be converted to a list, but the performance advantage is lost as the list has to be stored in memory.*



Example of a generator expression within a function:

*Generator comprehension ‘gen’ returns a generator object that can be iterated over to compare each value in the matrix to the target value. The search\_in\_matrix() function returns the position of the target if found, otherwise [-1, -1] is returned.*



Conclusion:

* Generators are much easier to implement than iterator classes
* They are more efficient in terms of memory (particularly useful for large data sets)
* They can be used to represent an infinite stream of data as only one item is produced at a time
* Multiple generators can be used to pipeline output of functions clearly and efficiently

1. **Decorators – A page for useful (or potentially abusive?) decorator ideas. What is the return type of the decorator?**

Pros:

* you can make an extension for class-methods without inheritance
* you could mix inheritance with decorators
* you still have the same class-object.

Cons:

* all methods needs to be initialized and re-assigned at init
* for each method you need to create another method in decorator (not flexible enough)
* decorator is limited to this class methods - can not be reused at different class with different methods.