**Q1**

**Encapsulation**

OOP encapsulation in Kotlin unlike Python is enforced and has some fine grained levels (scope modifiers/keywords) which are the following:

**private** — With a top level element it is not accessible outside the Kotlin file it is defined in, elements defined in a class/interface (eg properties, functions) are only accessible in the same place where they are defined.

**public** — Elements can be accessed anywhere.

**protected** — Same as private except sub classes can access class/interface elements (includes member properties and functions unless they are marked private), this encapsulation level isn’t supported on top level elements.

**internal** — Anything in the module can access elements defined in the same module.

A module in Kotlin is a set of Kotlin files which could be represented by a JAR file. By default all defined elements (not using a scope modifier) are public in scope just like Python.

**Advantages of encapsulation in Java are following −**

1. The encapsulated code is more flexible and easy to change with new requirements.

2. It prevents the other classes to access the private fields.

3. Encapsulation allows modifying implemented code without breaking other code who have implemented the code.

4. It keeps the data and codes safe from external inheritance. Thus, Encapsulation helps to achieve security.

5. It improves the maintainability of the application.

6. If you don’t define the setter method in the class then the fields can be made read-only.

7. If you don’t define the getter method in the class then the fields can be made write-only.

**Q2**

**Enum**

Enumerations in Kotlin are data types that hold a set of constants. Enums are defined by adding the modifier enum in front of a class as shown below. Yes, in Kotlin, Enums are classes.

Here are some important points about enum classes in kotlin.

1.Each enum constant is an object. Enum constants are separated by commas.

2.Each of the enum constants acts as separate instances of the class.

3.Enums are useful in enhancing the readability of your code since it assigns pre-defined names to constants.

4.Unlike classes, an instance of enum classes cannot be created using constructors.

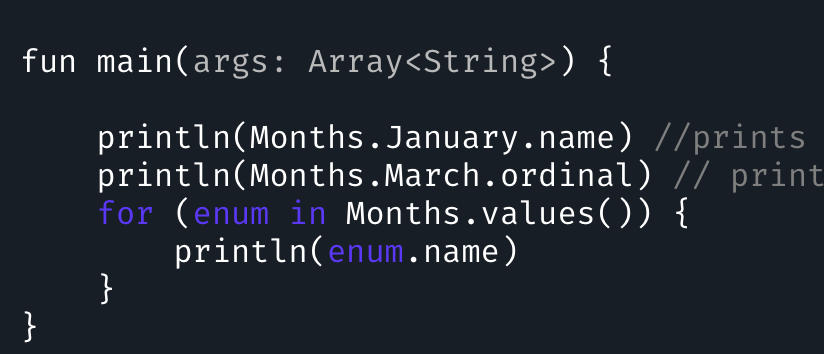
5.Hence, we can assert that enum classes are abstract.

**Q3**

**How to use ​enum​ in Kotlin?**

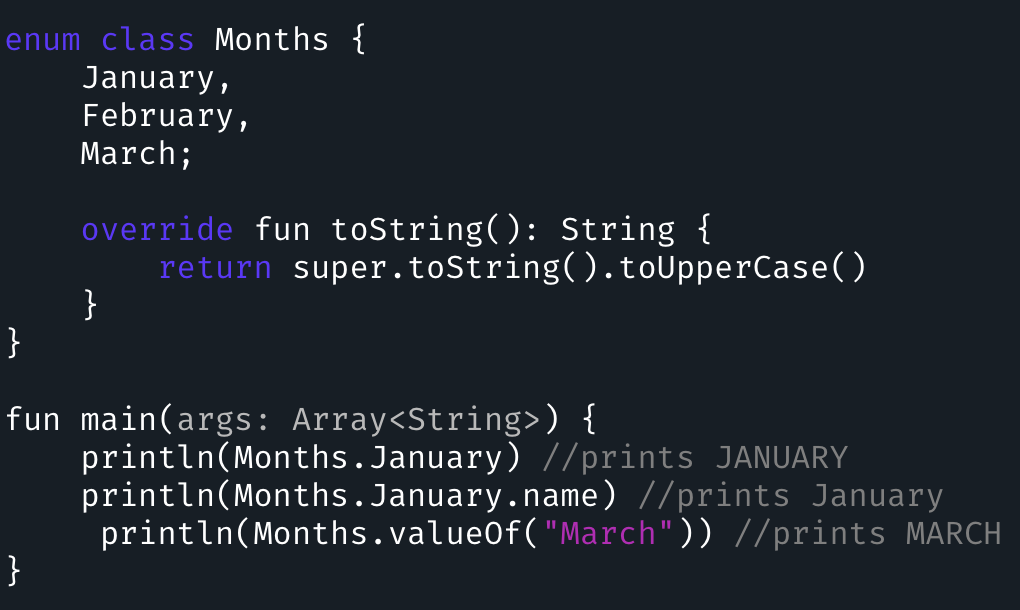
**Enum Properties**

Every enum constant has properties: name, ordinal to retrieve the name and position of the constant.



**Enum toString()**

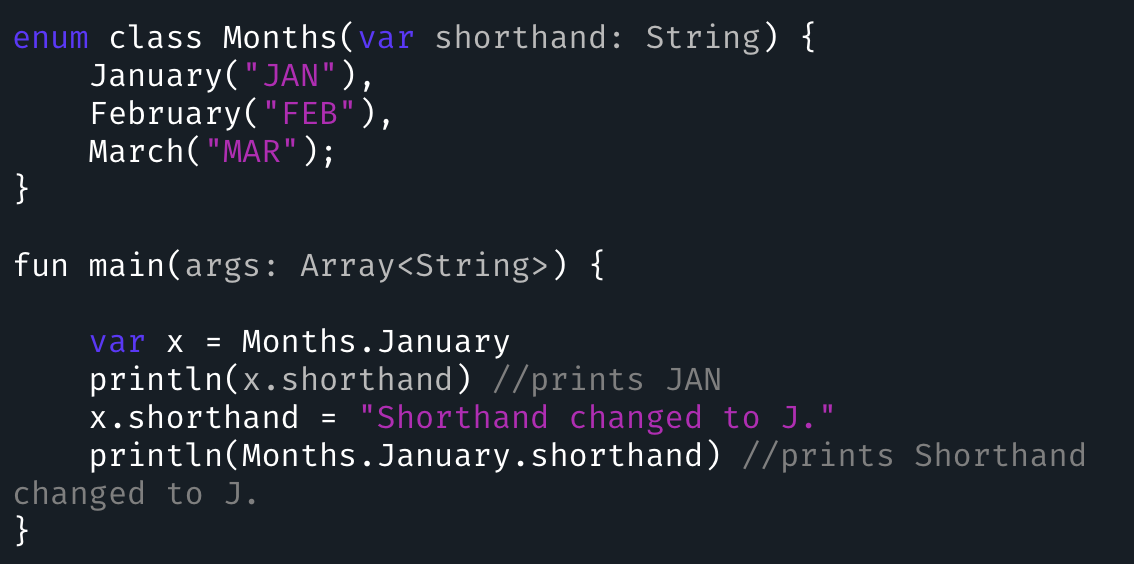
We can override the toString() function in the enum class as shown below.



Enums are classes. So besides defining the constants we can define other things as well that can be present in a class. To do so we need to first end the enum part with a semi colon.

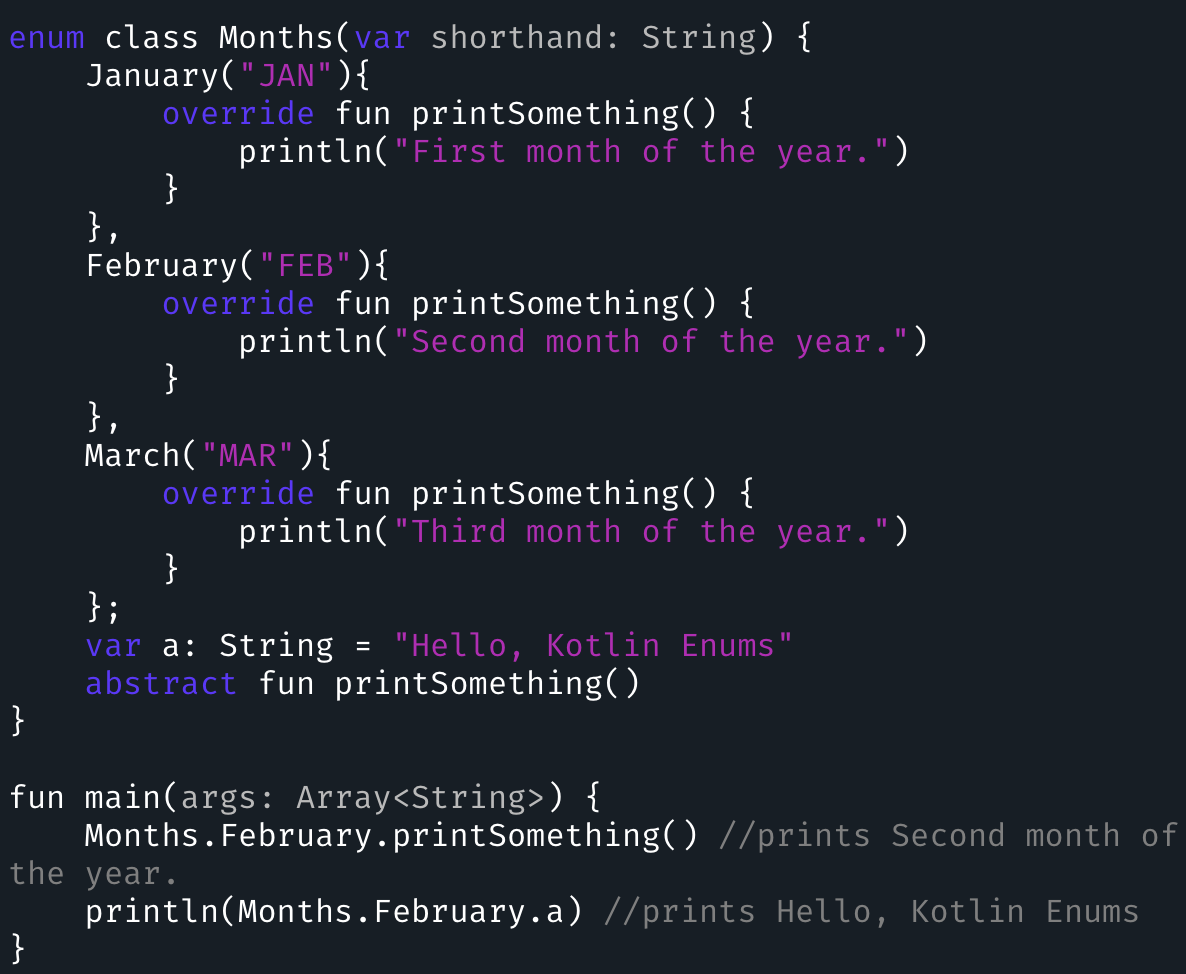
**Enum Initialisation**

Enum constants can be initialized using primary constructors as shown below.



**Enums as Anonymous classes**

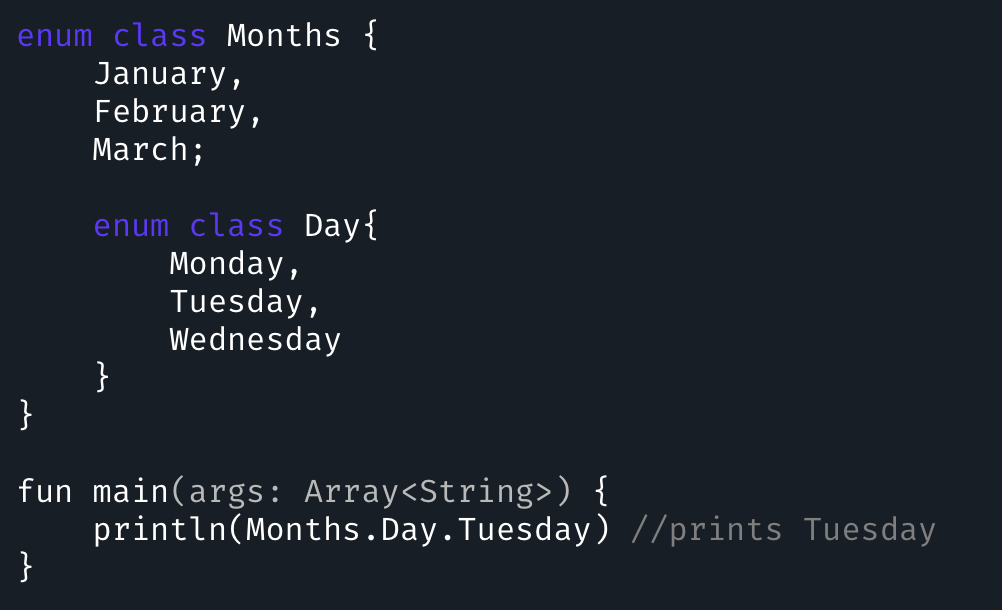
Enum constants can behave as anonymous classes be implementing their own functions along with overriding base functions from the class as shown below.



Each of the enum constants must override.

**Enum inside an Enum**

It’s possible to define another enum class in the current enum class as shown below.

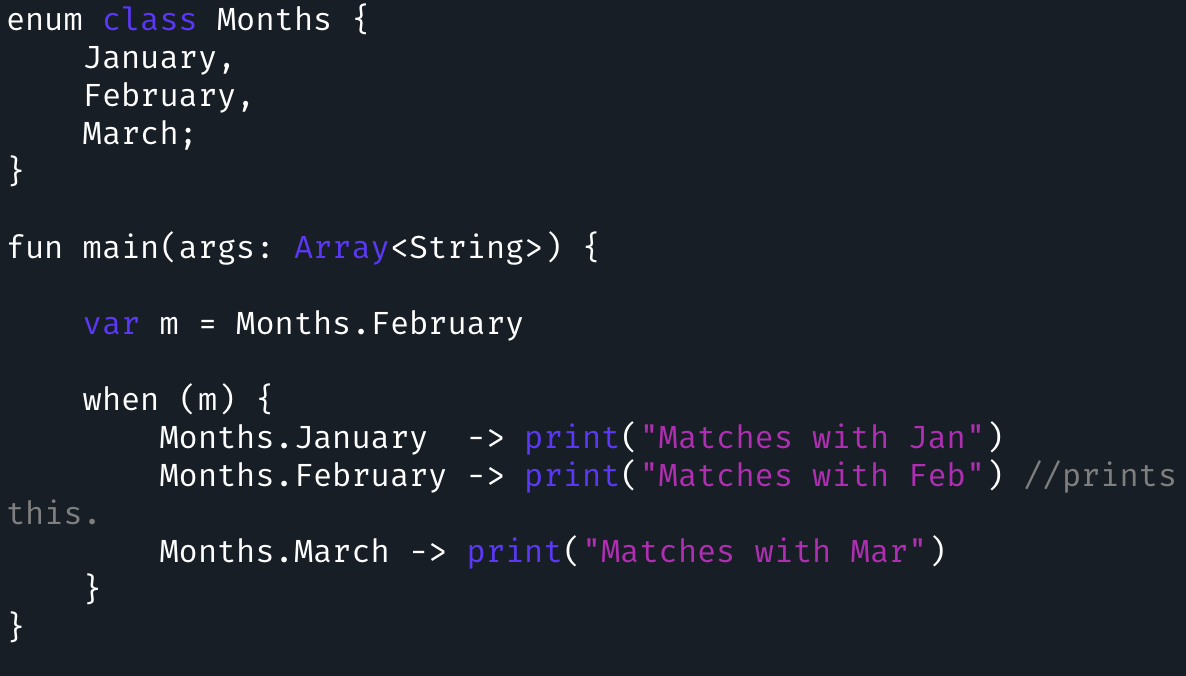


Only an enum class can invoke another one. The inner class Day cannot be invoked from the enum constants.

**Enums and when**

when is Kotlin equivalent of switch in Java. We’ve covered the workings of it in the Control Flow tutorial.

An example showing how when statements are used with enums is given below.



This brings an end to kotlin enum tutorial.

**Q4**

**Differences​ between ​LinearLayout​ and ​RelativeLayout​?**

**LinearLayout**

A linear layout displays its views continue one after one, either vertically or horizontally. You need to specify orientation to define whether layout is vertical or horizontal.

1.Weight: It specifies how much space each view spans relative to others. For example, in an e-mail application, you can give less weight to ‘To’ and ‘Subject’, and more weight to ‘Message’.

2.Gravity: It defines placement of a view’s contents.

3.Layout Gravity: It defines the placement of the view itself.

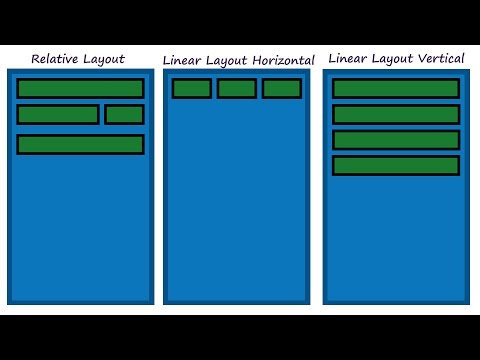
**RelativeLayout**

A relative layout displays its views relative to one another, so order is not that important. You can define the top most view at the end of the layout and provide details to show it on top left. Here are some points which will help you to understand about RelativeLayout.

1.Position relative to screen: You can align a view relative to screen using alignParentTop, centerHorizontal etc.

2.Position relative to other views: You can align a view relative to another view using above, below, toLeftOf etc.

3.Margins: You can provide margins using marginTop, marginLeft etc.



**Q5**

**The benefits of ​ConstraintLayout​**

1.Constraint Layout has dual power of both Relative Layout as well as Linear layout: Set relative positions of views ( like Relative layout ) and also set weights for dynamic UI (which was only possible in Linear Layout).

2.A very powerful use is grouping of elements by forming a chain. This way we can form a group of views which as a whole can be placed in a desired way without adding another layer of hierarchy just to form another group of views.

3.In addition to weights, we can apply horizontal and vertical bias which is nothing but the percentage of displacement from the centre. ( bias of 0.5 means centrally aligned. Any value less or more means corresponding movement in the respective direction ) .

4.Another very important feature is that it respects and provides the functionality to handle the GONE views so that layouts do not break if some view is set to GONE through java code. More can be found here:

5.Provides power of automatic constraint applying by the use of Blue print and Visual Editor tool which makes it easy to design a page.

All these features lead to flattening of the view hierarchy which improves performance and also helps in making responsive and dynamic UI which can more easily adapt to different screen size and density.

在開發過程中經常能遇到一些複雜的UI，可能會出現佈局巢狀過多的問題，巢狀得越多，裝置繪製檢視所需的時間和計算功耗也就越多。簡單舉個例子：

假設現在要寫一個這樣的佈局，可能有人會這麼寫：

首先是一個垂直的LinearLayout，裡面放兩個水平的LinearLayout，然後在水平的LinearLayout裡面放TextView。這樣的寫法就嵌套了兩層LinearLayout。

有些人考慮到了巢狀佈局帶來的風險，所以用一個RelativeLayout來裝下所有的控制元件。那麼問題來了，既然用RelativeLayout可以解決問題，為什麼還要使用ConstraintLayout呢？因為ConstraintLayout使用起來比RelativeLayout更靈活，效能更出色！還有一點就是ConstraintLayout可以按照比例約束控制元件位置和尺寸，能夠更好地適配螢幕大小不同的機型。