General advices:

1) If you see XML code in the question – its there for a reason
2) Query parameter is what comes after ? mark, like ?code=SFO
3) URI parameter is what comes after endpoint with /, like endpoint/5
4) JSON, XML, Java **are NOT** data types!! These are **data formats**
5) Data types – integer (or Number), String, Object, Array
6) When you import RAML in your Anypoint studio, the REST connector will generate as many **flows**, as many **methods** in your RAML you have (1 method = 1 flow)
7) Always pay attention to **batch size** in batch jobs XML code!
8) Always pay attention if **error-mapping** is present in XML code!
9) Read **carefully** what is asked in the question (example – what is last message logged? - you are asked not about total output, but about the very last message)
10) Exchange is **not** an asset
11) If you debug and and set breakpoint on processor – it will stop at this processor **before** executing it
12) Usually when mentioned "unique ID" or "unique parameter" in request – **URI** parameter is meant
13) Everything you write in literal mode is considered as a String. To make reference to, for example, payload, you should write #[payload]. Otherwise it will be just string "payload"
14) Empty string "", empty array [], empty object {} are **not null values.** They are still string, array and object, even the empty ones.
15) If you get BUILD FAILURE for not having dependency, simply install it
16) Via Flow Reference, everything (payload, variables, attributes) is transfered
17) Via HTTP Request, only payload is transfered and **exisiting attributes are overwritten**
18) 3 steps of creating modern API – Design, Build, Deploy/Manage
19) If you set payload as 10, it will be String "10". To make it as Number, write '10 as Number' in expression mode
20) Watermarks are set on column with **unique** values (usually IDs)
21) Synonyms: Single-threaded, Sequential, Synchronous
22) Synonyms: Multi-threaded, Parallel, Asynchronous
23) Mule event – mule message (payload and attributes) and variables
24) Scatter-Gather takes as much time, as the **longest** route
25) To create HTTP Listener or Request config, you need to specify: **Protocol, Port, Host**
26) HTTP configs with same prameters could be used for multiple requests/listeners
27) Always pay attention to what happens in the DataWeave script if you have XML code screenshot
28) Target variables are used to preserve some value for it NOT to be overwritten by next operation
29) XML objects always require root element
30) WSC (Web service consumer) is used in SOAP apis
31) WSDL files used in SOAP apis
32) Scaling vertically = changing worker size and keeping same worker amount
33) Scaling horizontally = changing worker amount and keeping same worker size
34) Scaling horizontally = distributing workload across workers
35) ObjectStore is used to preserve data across whole app
36) If you get error for incorrect driver, install the correct one

37) Concatenation of **String** and **Number** results in **String**
38) Dependencies are stored in **pom.xml**
39) Metadata is stored in **application-types.xml**

Error Handling:

1) If error occurs – flow execution **stops** (with only 1 exception – if error was thrown inside 'Try' scope and handled there right away with On Error Continue)
2) On Error Continue **deletes** whole error object at the end of error handler scope (kinda 'throws it under the carpet, like nothing happened')
3) On Error Propagate throws error 'like a hot potato' back to outer level
4) Always pay attention to the error type – error handler will execute only if error types match
5) If you have several Error Handlers – **the first true** one (first with matching error type) will fire, even if all of them are true
6) Error handler with type 'ANY' – is always true
7) If you have error handlers in the flow and **none** of them are true – then global error handler is also **ignored** only for that flow, where the error was thrown
8) By **default (!)** On Error Continue returns status code **200** and **Payload** as message
9) By **default (!)** On Error Propagate returns status code **500** and **error.description** as message
10) Each validator has its own built-in error message (example: if you check non-empty payload with 'is null' validator, error message will be 'payload is not null')
11) Responses can be customized (both error response and successfull response)

Syntax:

1) * - means 'all'
2) To reference property from config.yaml, use **dollar sign** and **curly braces** separated with **single dot** - ${namespace.property}
3) To reference query parameter, type attrubutes.queryParams.parameterName
4) To reference URI parameter, type attrubutes.uriParams.parameterName
5) To reference header, type attrubutes.headers.headerName
6) You **cannot** concatenate (using ++ operator) **Object** and **String** – result will be **error**
7) To use function from library **without** a namespace (functionName(something)), import it like:

   Import * from dw::core::LibraryName

   or

   Import functionName from dw::core::LibraryName
8) To use function from library **with** a namespace (moduleName::functionName(something)), import it like:

   Import modules::moduleName
9) To reference property from config.yaml in dataweave script, use 'p' letter – ***host: p('http.host')***
10) When putting URI paramaters in listener path, use **curly braces**: endpoint/{parameter}
11) To reference variable, use **vars** keyword (vars.variableName)
12) You **can** compare Strings using <, >, ==, != operators. Comparison is happening digitwise ("2" will be > than "10", because digit '2' is bigger that digit '1'). If first digits are same, the next ones are compared ("15" is > than "1", because digit '5' is bigger than null (null – second digit of "1")).

Loops:

1) After For-each loop you get **original** payload (even if you modified it inside the loop)
2) After For-each loop you get **modified** variables (of course if you modified them inside the loop)
3) After batch-job loop you get **original** payload (even if you modified it inside the loop)
4) After batch-job loop you get **original** variables (even if you modified them inside the loop)
5) If you created variables inside batch job, they will be null after batch job (what happened in batch job, stayed in batch job!)
6) In batch job in On Complete section you can access statictics (successful and failed records) and **not** the data itself
7) Batch size means how many objects (or strings, numbers, etc.) the loop will take inside before starting to process something. For example, setting batchSize=2 means that loop will take 2 objects at the time and will proces them together. Same goes for batch aggregator.
8) Batch steps can have a filter on it. If object doesn't pass filter for one batch step, it still can get into another batch step
9) Batch aggreagtor accepts only successfull records. Failed ones are not considered.

RAML:

1) If you defined data format of the body in the RAML (like application/json), you should send request with exactly this format, otherwise you get **415 Unsupported media type** error
2) If you defined example in RAML – then you should strictly follow the structure when sending request (maintain same field names, data types)
3) To reference file in your RAML, use ***!include*** and use relative path of the file
4) When defining RAML, the sequence of how you write lines should be: endpoint -> URI parameter (if exists) -> method
5) After you defined RAML, next step is to publish to exchange, to be able to import it to studio.

Choice:

1) Only **one** route will be executed
2) The route with the **first true condition** will be executed (even if all of them are true)
3) If non of the routes have true conditions, **default** will be executed

DataWeave:

1) Attribute of an Object – what is written in the same line with Object definition
2) Property of an Object – what is written inside of Object after the definition
3) If you need to reference attribute (not property) of XML object in mapping – use @
   Example: <book genre=novel>; in mapping – genre: value.@genre
4) Subflows **cannot** be called using 'lookup' function
5) To define the function, use '**fun**' (not the 'function' keyword). When using function in script, no keywords needed.
6) Concatenation of 2 JSON objects returns one full JSON object, having all properties of 2 input objects.
7) Using lookup function, you dont need to mention word 'payload' after flow name

Data types:

1) Database SELECT returns **Array**. If query returned 0 rows – it will still return Array, however empty one – []
2) DataWeave map operation returns inputs **Array** and returns **Array**
3) DataWeave mapObject operation inputs **Object** and returns **Object**
4) Scatter-Gather returns **Object of Mule event Objects**
5) Flatten function returns **Array**
6) File List operqation returns **Array of Mule event objects**

Files:

1) After ‘File read’ operation – nothing happens to the file
2) If you write to a file, that doesnt exists, the file will be created

Deployment:

1) After applying SLA policies, you need to add headers to RAML and redeploy
2) Modules and dependencies are enough to deploy to cloudhub
3) Both sources and modules/dependencies needed to be able to import and deploy in Anypoint Studio