Basic level coding best practices

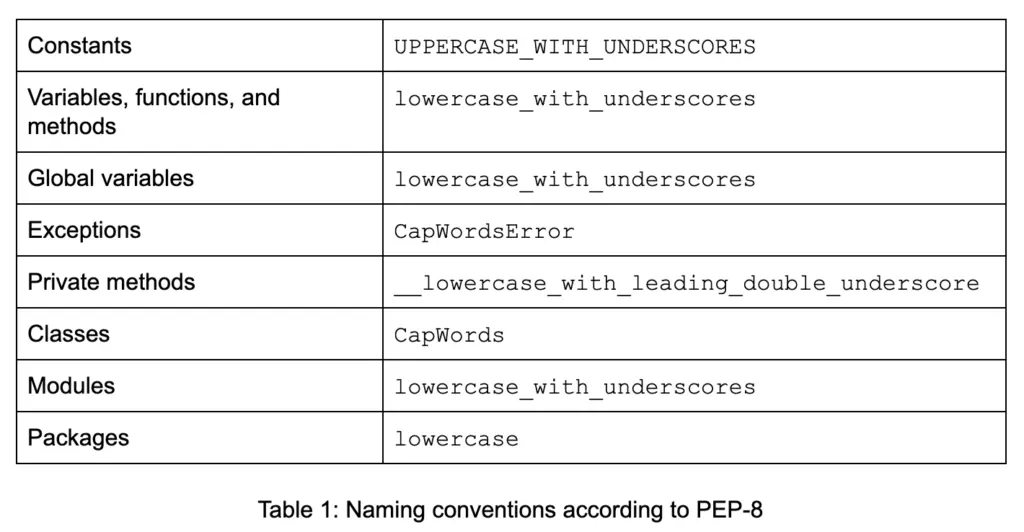
Maintained by: Architecture Group

Last Published: 29 November 2023

* ~~Author: Jamal Musatfa~~

## Naming conventions for Constants, functions, variables

* Prefix variable types like boolean to make them more readable - i.e "is" or "has" - (is\_enabled, has\_value etc.)
* Use plurals for arrays. For example, book\_names versus book\_name
* Describe numerical variables. Instead of just age consider avg\_age or min\_age depending on the purpose of the variable.



## Notebook Structure

* Poorly organized notebooks can be hard to read.
* Building a plan for your notebook before you start is a good idea.
* Establish the purpose of your notebook.
* In cases where there isn't a single straight forward goal, consider splitting work into multiple notebooks and creating a single master
* Use sections to help construct the right order of your notebook.
* Include a title, preamble, table of contents, conclusion and reference any sources

## Notebook coding styles

* Use 4 spaces per indentation level.
* Limit all lines to a maximum of 79 characters.
* Imports should usually be on separate lines
* Imports are always put at the top of the file, just after any module comments and docstrings.
* Absolute imports are recommended
* Wildcard imports *(from <module> import \*)* should be avoided
* Pick one rule for String Quotes and stick to it.

## Comments

* Comments should be complete sentences.
* Always make a priority of keeping the comments up-to-date when the code changes!
* Comments that contradict the code are worse than no comments.

## General guidelines & Tips

### Repeated code

Replace repeated code with a function e.g:

*import requests*

*def fetch\_and\_process\_data(api\_url):*

*# Fetch data from the API*

*response = requests.get(api\_url)*

*# Check if the request was successful (status code 200)*

*if response.status\_code == 200:*

*# Process the data (assuming JSON response in this example)*

*data = response.json()*

*# Perform some actions on the processed data*

*processed\_data = process\_data(data)*

*perform\_actions(processed\_data)*

*else:*

*print(f"Failed to fetch data. Status code: {response.status\_code}")*

*def process\_data(data):*

*# Placeholder for data processing logic*

*# You can replace this with your actual data processing code*

*return data*

*def perform\_actions(processed\_data):*

*# Placeholder for actions to be performed on processed data*

*# You can replace this with your actual action code*

*print(f"Performing actions with processed data: {processed\_data}")*

*# Example usage*

*api\_url1 = "https://api.example.com/data1"*

*fetch\_and\_process\_data(api\_url1)*

*api\_url2 = "https://api.example.com/data2"*

*fetch\_and\_process\_data(api\_url2)*

*api\_url3 = "https://api.example.com/data3"*

*fetch\_and\_process\_data(api\_url3)*

### Connection information

In case if the notebook is connecting with any blob or SharePoint location, the actual connection string should be written only once in the whole notebook. E.g.

*conn = sp.connect("https://pgone.sharepoint.com", "muneeb.mm@pg.com", "xxxxxx") # Authorizing with the sharepoint account*

*#better*

*sp\_location = "*[*https://pgone.sharepoint.com*](https://pgone.sharepoint.com)*" # only once in your whole notebook*

*conn = sp.connect(sp\_location, "muneeb.mm@pg.com", "xxxxxx") # Authorizing with the sharepoint account*

### Secure information

Usernames, Passwords, Hostnames should not be maintained in direct python file or notebook. Sensitive information should be managed in secure vault, and it should be referenced in python file or notebook using keys.

*#Wrong*

*conn = sp.connect("https://pgone.sharepoint.com", "muneeb.mm@pg.com", "xxxxxx") # Authorizing with the sharepoint account*

Instead we should write as:

*#Correct*

*dbutils.secrets.get(scope = “databricks-scope”, key = “sp\_username”)*

*dbutils.secrets.get(scope = “databricks-scope”, key = “sp\_pass”)*

#Correct

*STORAGEACCOUNTNAME= "blobxxxx01"*

*STORAGEACCOUNTKEY = dbutils.secrets.get(scope = "databricks-scope", key = "blobstoragekey")*

Boolean Comparison: Booleans are already Booleans. They don’t need comparisons.

*# Correct:*

*if is\_active\_customer:*

*do\_something()*

*# Wrong:*

*if is\_active\_customer == True:*

*do\_something()*

### Lazy evaluation

—--------

*# Build an execution plan.*

*# This returns in less than a second but does no work*

df2 = (df

.join(...)

.select(...)

.filter(...)

)

*# Now run the execution plan to get results*

df2.display()

*# Unfortunately this will run the plan again, including filtering, joining, etc*

df2.display()

*# So will this…*

df2.count()

—------

How do we avoid the extra computation? The answer is pretty straightforward: save computed results you will reuse.

*# Build an execution plan.*

*# This returns in less than a second but does no work*

df2 = (df

.join(...)

.select(...)

.filter(...)

)

*# save it*

df2.write.save(path)

*# load it back in*

df3 = spark.read.load(path)

*# now use it*

df3.display()

*# this is not doing any extra computation anymore. No joins, filtering, etc. It's already done and saved.*

df3.display()

*# nor is this*

df3.count()

—------

### Directly View the content of a file

Here is an example to directly view the content of a file by using the “display” function and dbutils utility:

*display(dbutils.fs.head("/path/to/file.csv"))*

In this example, the “display" function is used to display the contents of the CSV file located at “/path/to/file.csv“. The dbutils.fs.head() function is used to read the first few lines of the file. If we use Dataframe API to read the file and view it then that will be an expensive operation.

### Notebook Chaining

It is always a good practice to include all the repeatedly used operations such as read/write on Data Lake, SQL Database, etc., in one generic Notebook. The same Notebook can be used to set the Spark configurations, mounting ADLS path to DBFS, fetching the secrets from secret scope, etc.

For using the operations defined in the generic Notebook from other notebooks, it should be invoked using the “run” command.

%run "/path\_to/Notebook A"

%run "/path\_to/Notebook B"