m01_workshop2_guided

August 7, 2025

1 M01 Python Engineering - Workshop 2

Each challenge includes a stretch option for those who are already confident or looking for an extra challenge. These stretch tasks are completely optional, so don't worry if you don't get to them. The most important thing is to focus on the core challenge, as that's where the key learning happens. If you do feel ready to go further, give the stretch a try!

1.1 Challenge: Refactor a Python script into modules

Goal: Refactor Python code into well-structured modules using functions and demonstrate importing and running code.

- Create a folder named `my_utils`
- Within `my_utils` create two blank Python scripts `main.py` and `arith_utils.py`

```
my_utils/                  # Project folder
  main.py                 # Main script (entry point)
  arith_utils.py          # Module: for math-related functions
```

- Split the code given below into:
  - `arith_utils.py`: a module for the math-related functions `factorial` and `is_prime`
  - `main.py`: a main script that imports and uses the functions from `arith_utils.py`
- Run `main.py` from your CLI or IDE (inside the `my_utils` folder)
- *Stretch:*
  - Add a third function to `arith_utils.py` called `is_even` that returns `True` if a number is even.
  - Update `main.py` to also print whether each number is even.

[23]:
```python
def factorial(n):
    """
    Calculate the factorial of a given number.

    Args:
        n (int): A non-negative integer.

    Returns:
        int: The factorial of n (i.e., n!).
```

```
    """
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result


def is_prime(n):
    """
    Check if a number is a prime number.

    Args:
        n (int): The number to check.

    Returns:
        bool: True if n is prime, False otherwise.
    """
    if n < 2:
        return False
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            return False
    return True

# List of numbers to analyze
numbers = [3, 5, 10]

# Loop through each number and print:
# the factorial of the number
# whether it is a prime number
for n in numbers:
    print(f"{n}! = {factorial(n)}, Prime? {is_prime(n)}")
```

```
3! = 6, Prime? True
5! = 120, Prime? True
10! = 3628800, Prime? False
```

1.2 Challenge: Setup a clean Python environment

Goal: Practice setting up an isolated Python environment and managing dependencies for reliable, reproducible AI projects.

- From the CLI, run the following code to create a virtual environment called `my_env`:

```
python -m venv my_env
```

- From the CLI, run the following code to activate the virtual environment called `my_env`:

```
source my_env/bin/activate
```

- Install `pandas` in your environment (`pip install pandas`)

- Create a short script that:
  - Imports `pandas` as `pd`
  - Reads the provided CSV file `reviews.csv` with `pd.read_csv()`
  - Prints the first few rows with `.head()`
- Freeze your environment to `requirements.txt` (hint: `pip freeze > requirements.txt`)
- *Stretch:*
  - Add a `README.md` file to your project folder. It should include:
    * A brief description of the project and its purpose
    * Clear setup instructions (how to activate the environment and install dependencies)
    * How to run the script
    * What the script does and what output to expect

1.3 Challenge: Apply OOP in Python

Goal: Practice using classes in Python to encapsulate data and behavior.

1. Define a class called `Review` that represents a text review with:
   - An `__init__` method that takes two parameters: `text` and `label`
     - Use `def __init__(self, text, label):` to set up the constructor
     - Inside `__init__`, store the values using `self.text = text` and `self.label = label`
   - A method `is_positive()` that returns `True` if the label is `"positive"` (case-insensitive), otherwise `False`
     - Use `def is_positive(self):` to define the method
     - Check if the label is "positive" using: `return self.label.lower() == "positive"`
2. Create two objects using your class — one with a positive label, one with a negative label e.g. `review1 = Review("Great!", "positive")`
`review2 = Review("Bad!", "negative")`
3. Print out the review text and whether it's positive or not using `.is_positive()` e.g. `print(review1.text, review1.is_positive())`
4. *Stretch:*
   - Add a method `word_count()` that returns the number of words in the review text
     - Use `def word_count(self):` to define the method
     - Use `.split()` on `self.text` to get words: `words = self.text.split()`
     - Return the number of words with `return len(words)`
   - Use it to print the word count for both reviews

```
[ ]:
```

1.4 Challenge: Add Error Handling to a Script

Goal: Practice using `try` / `except` blocks to catch and handle common errors in Python.

Review the common built-in exceptions in Python in the table below (https://docs.python.org/3/library/exceptions.html)

Exception	Description
`SyntaxError`	Raised when the parser encounters a syntax error
`NameError`	Raised when a variable is not found in the local or global scope
`TypeError`	Raised when an operation is applied to an object of inappropriate type
`ValueError`	Raised when a function receives an argument of the right type but wrong value
`IndexError`	Raised when a sequence index is out of range
`KeyError`	Raised when a dictionary key is not found
`AttributeError`	Raised when an attribute reference or assignment fails
`ImportError`	Raised when an import statement fails to find the module
`ModuleNotFoundError`	Subclass of `ImportError` for when the module itself cannot be found
`ZeroDivisionError`	Raised when a number is divided by zero
`FileNotFoundError`	Raised when a file or directory is requested but doesn't exist
`IOError`	Raised when an input/output operation fails
`RuntimeError`	Raised when an error is detected that doesn't fall in any other category
`NotImplementedError`	Raised to indicate that a method or function hasn't been implemented yet
`AssertionError`	Raised when an `assert` statement fails

Review the following code which asks the user for a number, divides 100 by it and prints the result:

```
user_input = input("Enter a number: ")
number = int(user_input)
result = 100 / number
print("Result is:", result)
```

Refactor the code by adding error handling to:

- Catch invalid input e.g. if the user types `"hello"` instead of a number (hint: `ValueError`)
- Prevent a crash if the user types `0` (hint: `ZeroDivisionError`)
- Display a helpful message in each case
- *Stretch:*
  - Add a `finally:` block that prints `"Finished processing input."` whether or not there was an error.
  - Wrap this in a function called `safe_divide()` and call it.

[31]:
```
user_input = input("Enter a number: ")
number = int(user_input)
result = 100 / number
print("Result is:", result)
```

```
Enter a number:  50
```

```
Result is: 2.0
```

END