# Workshop 3: Functions and modules

**FIE463: Numerical Methods in Macroeconomics and Finance using Python**

## Richard Foltyn
### *NHH Norwegian School of Economics*

## January 30, 2025

See GitHub repository for notebooks and data:

https://github.com/richardfoltyn/FIE463-V25

## Exercise 1: Standard deviation of a sequence of numbers

The standard deviation $\sigma$ characterizes the dispersion of a sequence of data $(x_1, x_2, \ldots, x_N)$ around its mean $\overline{x}$. It is computed as the square root of the variance $\sigma^2$, defined as

$$\sigma^2 = \frac{1}{N} \sum_{i=1}^{N} \left( x_i - \overline{x} \right)^2$$

where $N$ is the number of elements (we ignore the degrees-of-freedom correction), and the mean $\overline{x}$ is defined as

$$\overline{x} = \frac{1}{N} \sum_{i=1}^{N} x_i$$

The above formula for the variance can be rewritten as

$$\sigma^2 = \left( \frac{1}{N} \sum_{i=1}^{N} x_i^2 \right) - \overline{x}^2$$

This suggests the following algorithm to compute the standard deviation:

1. Compute the mean $\overline{x} = \frac{1}{N} \sum_{i=1}^{N} x_i$

2. Compute the mean of squares $S = \frac{1}{N} \sum_{i=1}^{N} x_i^2$

3. Compute the variance $\sigma^2 = S - \overline{x}^2$

4. Compute the standard deviation $\sigma = \sqrt{\sigma^2}$

In this exercise, you are asked to implement the above algorithm and compare your function with NumPy's implementation `np.std()`.

1. Create a module `my_stats.py` and add the function

   ```python
   def my_std(x):
       """
       Compute and return the standard deviation of the sequence x.
       """
   ```

   which implements the above algorithm to compute the standard deviation of a given sequence `x` (this could be a tuple, list, array, etc.). Your implementation should *only use built-in functions* such as `len()`, `sum()`, and `sqrt()` from the `math` module.

2. Import this function into the Jupyter notebook. Using an array of 11 elements which are uniformly spaced on the interval $[0, 10]$, confirm that your function returns the same value as `np.std()`.

3. Benchmark your implementation against `np.std()` for three different arrays with 11, 101, and 10001 elements which are uniformly spaced on the interval $[0, 10]$.

   *Hint:* Use the cell magic `%timeit` to time the execution of a statement.

## Exercise 2: Locating maximum values

In this exercise, you are asked to write a function that returns the position of the largest element from a given sequence (list, tuple, array, etc.).

1. Write a function `my_argmax()` that takes as argument a sequence and returns the (first) index where the maximum value is located. Only use built-in functionality in your implementation (no NumPy).

2. Create an array with 101 values constructed using the sine function,

   `arr = np.sin(np.linspace(0.0, np.pi, 101))`

   and use it to test your function.

3. Compare the result returned by your function to NumPy's implementation `np.argmax()`.

## Exercise 3: Two-period consumption-savings problem

This exercise asks you to find the utility-maximizing consumption levels using grid search, an algorithm that evaluates all possible alternatives from a given set (the "grid") to locate the maximum.

Consider the following standard consumption-savings problem over two periods with lifetime utility $U(c_1, c_2)$ given by

$$\max_{c_1,\, c_2} \quad U(c_1, c_2) = u(c_1) + \beta u(c_2)$$

$$\text{s.t.} \quad c_1 + \frac{c_2}{1+r} = w$$

$$c_1 \geq 0,\ c_2 \geq 0$$

where $\beta$ is the discount factor, $r$ is the interest rate, $w$ is initial wealth, $(c_1, c_2)$ is the optimal consumption allocation to be determined. The second line is the budget constraint which ensures that the chosen consumption bundle $(c_1, c_2)$ is feasible. The per-period CRRA utility function $u(c)$ is given by

$$u(c) = \begin{cases} \frac{c^{1-\gamma}}{1-\gamma} & \text{if } \gamma \neq 1 \\ \log(c) & \text{if } \gamma = 1 \end{cases}$$

where $\gamma$ is the coefficient of relative risk aversion (RRA) and $\log(\bullet)$ denotes the natural logarithm.

1. Write a function `util(x, gamma)` which evaluates the per-period utility $u(c)$ for a given consumption level $c$ and the parameter $\gamma$. Make sure to take into account the log case!

   *Hint:* You can use the `np.log()` function from NumPy to compute the natural logarithm.

2. Write a function `util_life(c_1, c_2, beta, gamma)` which uses `util()` from above to compute the lifetime utility $U(c_1, c_2)$ for given consumption levels $(c_1, c_2)$ and parameters.

3. Assume that $r = 0.04$, $\beta = 0.96$, $\gamma = 1$, and $w = 1$.

   - Create a candidate array (grid) of period-1 consumption levels with 100 grid points with are uniformly spaced on the on the interval $[10^{-5}, w]$.

   - Compute the implied array of period-2 consumption levels from the budget constraint.

- Given these candidate consumption levels, use the function `util_life()` you wrote earlier to evaluate lifetime utility for each bundle of consumption levels $(c_1, c_2)$.

4. Use the function `np.argmax()` to locale the index at which lifetime utility is maximized. Print the maximizing consumption levels $(c_1, c_2)$ as well as the associated maximized utility level.