Dynamic Modelling Course - TEP4290: Warm-up 7

The point of this exercise is for you to practice and familiarize yourself with importing modules and packages, something that you can learn about in the recommended what you have learned in the recommended video and notebook:

- https://www.youtube.com/watch?v=Uei2ILcxuPs and
- https://github.com/jakevdp/WhirlwindTourOfPython/blob/master/13-Modules-and-Packages.ipynb

You will do some simple imports here to get comforable with modules and packages in Python. The exercise is pass/fail.

Good luck!

Quick summary

What are modules, packages and libraries

Fundamentally, all three are resources of code that you can import to make use of functions and classes from that code. The difference is in the scope: a module is a single file, which you might just have written yourself, while a package is a collection of modules, and a library a collection of packages.

Third party modules/packages/libraries

While you can write your own code and import it to another file, you can also make use of code that other people published, typically in the form of packages and libraries like NumPy, Pandas or Matplotlib. While importing from your own modules requires to have a Python file (manually) available to import from, many are made publically available, so they can be installed through easy commands (or are even preinstalled) so you can import them more conveniently.

Installing modules only needs to be done once for your Python environment (but might be difficult on cloud based notebooks though). You can use 'pip install' directly in the notebook, or 'conda install' in the anaconda prompt if you run your notebooks locally.

Making your own modules

To make your own modules, you just need a Python (or notebook) file with some functions or classes in the same folder as the current file. You can then import these function/class

definitions the same way as you would for third party modules.

Imports

You fundamentally have the option to

- 1. import all definitions of another module for direct use,
- 2. to import the module so you can call the functions by calling module.function()
- 3. to rename the module for the import,
- 4. importing certain functions only. See the examples below:

In [1]:
```
#option 2
import numpy
numpy.zeros(5)
```

Out[1]:
```
array([0., 0., 0., 0., 0.])
```

In [2]:
```
#option 3 (most common for the standard libraries like numpy, pandas etc)
import numpy as np
np.zeros(5)
```

Out[2]:
```
array([0., 0., 0., 0., 0.])
```

In [3]:
```
#option 4
from numpy import zeros
zeros(5)
```

Out[3]:
```
array([0., 0., 0., 0., 0.])
```

Imports from jupyter notebooks look a bit differently, but are very convenient for your project. Since your jupyter notebook works somehow different than a regular python file, we can (and should) specify to only import the definitions (avoids running all the other commands that might be in the notebook). If you have the Jupyter notebook called WU7_external_functions in the same folder, the following code should work:

Hint: If you get the error message that no module ipynb was found, install the ipynb module first (see next section).

In [4]:
```
from ipynb.fs.defs.Warm_up_07_external_functions import example_function

example_function()
```

```
The import of the example function was successful!
```

Installing

While there are some standard libraries that most python installers already install for you, the majority of external python libraries you will have to install before using. While you can install libraries from local repositories, the easiest way to install common libraries is using

pythons inegration of "install". Python then automatically checks if you already have the package installed and what requirements a certain package might have, downloads and installs these for you.

Most of you will not have the ipynb module installed yet, so you can do it here (if you have problems, an alternative is the anaconda promt with "conda install [insert package name")-

In [5]:
```
#uncomment the next line to install and then comment once you installed the package
#%pip install ipynb
```

Best practice for importing

While imports can work in a variety of ways, there are some beste practices for how to structure your imports:

- Imports should usually be on separate lines
- Imports are always put at the top of the file, just after any module comments and docstrings, and before module globals and constants.
- Imports should be grouped in the following order:
  - Standard library imports.
  - Related third party imports.
  - Local application/library specific imports.
- You should put a blank line between each group of imports.

These are recommendations taken from the PEP 8 style guide for python, https://www.python.org/dev/peps/pep-0008/#fn-hi

Tasks

Complete the tasks outlined below to achieve the same output as you find in the original file. Use the specific method described if applicable.

Import...as from third party standard libraries

import numpy and pandas such that the following operations work:

In [6]:
```
import numpy as nmpy
import pandas as pnds


matrix = nmpy.eye(5)
print(matrix)
df = pnds.DataFrame(matrix)
print(df)
```

```
[[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]]
     0    1    2    3    4
0  1.0  0.0  0.0  0.0  0.0
1  0.0  1.0  0.0  0.0  0.0
2  0.0  0.0  1.0  0.0  0.0
3  0.0  0.0  0.0  1.0  0.0
4  0.0  0.0  0.0  0.0  1.0
```

In [7]:
```
import numpy as numericalOperationsPython
import pandas as AiluropodaMelanoleuca

matrix = numericalOperationsPython.eye(5)
print(matrix)
df = AiluropodaMelanoleuca.DataFrame(matrix)
print(df)
```

```
[[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]]
     0    1    2    3    4
0  1.0  0.0  0.0  0.0  0.0
1  0.0  1.0  0.0  0.0  0.0
2  0.0  0.0  1.0  0.0  0.0
3  0.0  0.0  0.0  1.0  0.0
4  0.0  0.0  0.0  0.0  1.0
```

Importing a cosine function

Often you will know what you want to do, but not how to do it in Python, so it makes sense to look for functions that you want use. In this task you will look up two ways to import a cosine function into python and use both in the second chunk (the first one ist just to set up, you do not need to change it)

Setup - not a task!

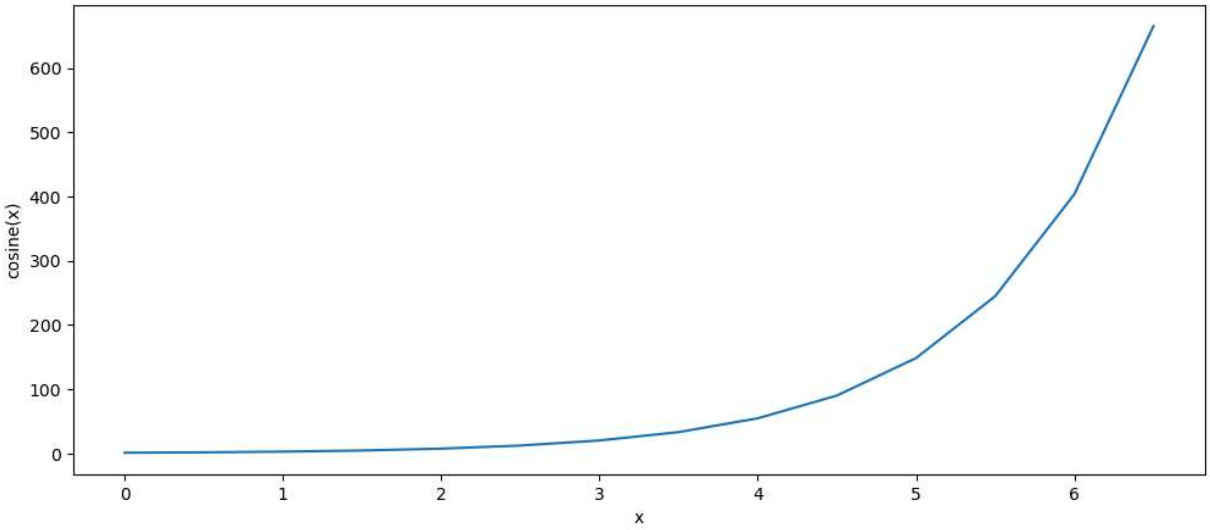
You don't need to understand precisely what the print_cosine function does, just that it prints the cosine between 0 and 2 pi if you hand it a cosine function. The exponential function is just an example!

In [8]:
```
import matplotlib.pyplot as plt # for plotting
import numpy as np #for the e function

def print_cosine(cosine):
    '''Prints cosine between 0 and two pi'''
    plt.figure(figsize = (12,5))
    numbers = [0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5, 5.5, 6, 6.5]
    y = []
```

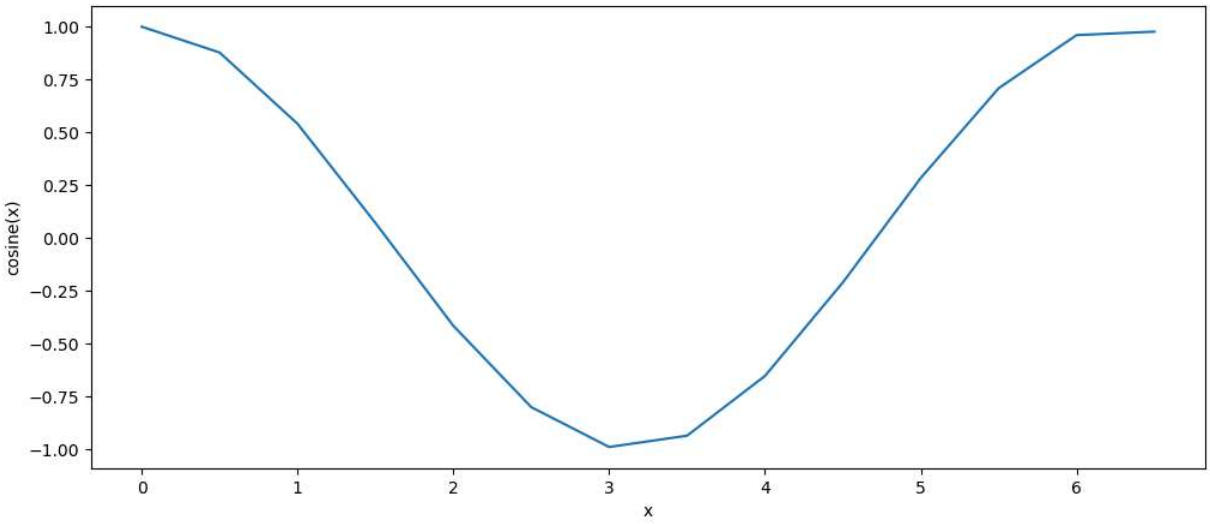
```
    for x in numbers:
        y.append(cosine(x))
    plt.plot(numbers, y)
    plt.xlabel('x')
    plt.ylabel('cosine(x)')

print_cosine(np.exp) #also works with not cosine functions...
```



now import a cosine function and use it in the print_cosine function to make the plots below

In [9]:
```
import math
print_cosine(math.cos)
```



Import from another Python file

Define the function at the top of the WU7_py_file, name it after yourselve ('name_greets', arguments: person) and import it to this file

In [10]:
```
from Warm_up_07 import Yiwen_greets
Yiwen_greets('Marceau')
```

```
Hello Marceau. How are you?
```

Import entire module

Import the module from the Warm_up07_external_functions notebook as wu to this notebook and use the functions defined there to finish the gameshow program.

In [11]:
```python
import ipynb.fs.defs.Warm_up_07_external_functions as wu
def gameshow():
    print("Welcome to the CodeInPlace Game Show")
    print("Pick a door and win a prize")
    print("-----------------------------------")

    door = wu.get_door()
    print("Door:", door)

    prize = wu.compute_prize(door)

    print('You win ' + str(prize) + ' treats')

gameshow()
```

```
Welcome to the CodeInPlace Game Show
Pick a door and win a prize
-----------------------------------
Invalid door!
Door: 1
You win 2 treats
```