EE2703 - Week 1

Nitin Chandrachoodan <nitin@ee.iitm.ac.in>

January 19, 2023

1 Introduction - Delete

Welcome to EE2703 Applied Programming Lab! The purpose of the lab is to use programming concepts to implement certain concepts that you would have (or are likely to) come across in the broad domain of Electrical Engineering.

We will be using Python as the programming language of choice for this course. The pros and cons of this choice are discussed in more detail in the presentation for the course. This Python notebook on the other hand is meant as an exercise to learn some of the operational issues regarding assignment submission and evaluation.

1.1 General approach

Each week, the problem statement will be given out as a PDF file as well as a Python notebook. This notebook will contain some template or stub code, as well as some parts that are to be implemented by you.

Certain sections of the notebook will also have a "Delete" marked in the section heading. These sections **should be removed by you** before submitting your solution.

For all other sections, you need to modify the text so that it is a readable document reporting your work, and add any code and results that are needed to generate a final report. Most sections will have a description of the problem statement in *quoted* form - this part is to be removed and replaced with a description of what you have done.

Your report will be graded based on the quality:

- correctness and completeness
- aesthetics - visual quality

The rest of this document consists of assignments you need to complete and submit on Moodle.

1.2 Notebook or Standalone?

These assignments are created as Python notebooks, and are tested mainly on the lab server https://jup.dev.iitm.ac.in/. The easiest way to proceed would therefore be to upload the notebook to your folder on this server, and then edit as required. Finally, you can generate a PDF from the notebook for submission.

Alternatively, you can also choose to use other approaches:

- Use the Visual Studio Code (VSCode) editor - this has good support for editing and running Python notebooks, and can be installed locally on your laptop

- Run a local Python jupyter server - instructions for this are readily available by searching on the internet
- Certain other editors like PyCharm *may* also support importing and editing notebooks.

None of the other techniques are officially supported. You are welcome to add documentation on how to use them on the Moodle forum, but we (instructor and TAs) may not be able to help with any problems you face.

1.3 Documentation

If you use the Python notebooks on the lab server, you should be able to directly "Save and export as" PDF, which will run LaTeX to generate neatly formatted documents.

However, if you choose one of the other approaches, you are responsible for generating your own documentation. Future lab sessions may cover the basics of LaTeX for documentation, but in the meantime you are encouraged to learn on your own.

Note that PDF generated by MS Word, Google docs, LibreOffice etc. are usually of relatively poor quality when equations and plots are to be included, and should not be used without a clear understanding of what is expected.

1.4 Assignment submission

You need to submit a single ZIP file for each assignment. That file in turn should contain the following:

- Your python notebook(s) and/or other Python files that contain your code
- A PDF report that contains the final documentation of your work
- Any other figures, data, or supporting files required to generate and/or run your code

2 Document metadata

> *Problem statement: modify this document so that the author name reflects your name and roll number. Explain the changes you needed to make here. If you use other approaches such as LaTeX to generate the PDF, explain the differences between the notebook approach and what you have used.*

3 Basic Data Types

Here we have a series of small problems involving various basic data types in Python. You are required to complete the code where required, and give *brief* explanations of your answers. Remember that the documentation and explanation is as important as the answer.

For each of the following cells, first execute them, and then give a brief explanation of why the answer comes out to be the way it does. If there is an error during execution of the cell, explain how you fixed it. **Add a new cell of type Markdown with the explanation** after the corresponding cell. If you are using plain Python, add suitable comments after each line and explain this in the documentation (clearly you would be better off using Notebooks here).

3.1 Numerical types

[ ]:
```
print(12 / 5)
```

[ ]:
```
print(12 // 5)
```

[ ]:
```
a=b=10
print(a,b,a/b)
```

3.2 Strings and related operations

[ ]:
```
a = "Hello "
print(a)
```

[ ]:
```
print(a+b)  # Output should contain "Hello 10"
```

[ ]:
```
# Print out a line of 40 '-' signs (to look like one long line)
# Then print the number 42 so that it is right justified to the end of
# the above line
# Then print one more line of length 40, but with the pattern '*-*-*-'
```

[ ]:
```
print(f"The variable 'a' has the value {a} and 'b' has the value {b:>10}")
```

[ ]:
```
# Create a list of dictionaries where each entry in the list has two keys:
# - id: this will be the ID number of a course, for example 'EE2703'
# - name: this will be the name, for example 'Applied Programming Lab'
# Add 3 entries:
# EE2703 -> Applied Programming Lab
# EE2003 -> Computer Organization
# EE5311 -> Digital IC Design
# Then print out the entries in a neatly formatted table where the
# ID number is left justified
# to 10 spaces and the name is right justified to 40 spaces.
# That is it should look like:

# EE2703                     Applied Programming Lab
# EE2003                       Computer Organization
# EE5131                           Digital IC Design
```

4 Functions for general manipulation

[ ]:
```
# Write a function with name 'twosc' that will take a single integer
# as input, and print out the binary representation of the number
# as output.  The function should take one other optional parameter N
# which represents the number of bits.  The final result should always
# contain N characters as output (either 0 or 1) and should use
# two's complement to represent the number if it is negative.
```

```
# Examples:
# twosc(10): 0000000000001010
# twosc(-10): 1111111111110110
# twosc(-20, 8): 11101100
#
# Use only functions from the Python standard library to do this.
def twosc(x, N=16):
    pass
```

5 List comprehensions and decorators

[ ]:
```
# Explain the output you see below
[x*x for x in range(10) if x%2 == 0]
```

[ ]:
```
# Explain the output you see below
matrix = [[1,2,3], [4,5,6], [7,8,9]]
[v for row in matrix for v in row]
```

[ ]:
```
# Define a function `is_prime(x)` that will return True if a number
# is prime, or False otherwise.
# Use it to write a one-line statement that will print all
# prime numbers between 1 and 100
```

[ ]:
```
# Explain the output below
def f1(x):
    return "happy " + x
def f2(f):
    def wrapper(*args, **kwargs):
        return "Hello " + f(*args, **kwargs) + " world"
    return wrapper
f3 = f2(f1)
print(f3("flappy"))
```

[ ]:
```
# Explain the output below
@f2
def f4(x):
    return "nappy " + x

print(f4("flappy"))
```

6 File IO

[ ]:
```
# Write a function to generate prime numbers from 1 to N (input)
# and write them to a file (second argument).  You can reuse the prime
# detection function written earlier.
def write_primes(N, filename):
```

```
    pass
```

7 Exceptions

[ ]:
```
# Write a function that takes in a number as input, and prints out
# whether it is a prime or not.  If the input is not an integer,
# print an appropriate error message.  Use exceptions to detect problems.
def check_prime(x):
    pass
x = input('Enter a number: ')
check_prime(x)
```

[ ]:
```

```