EE2703

Applied Programming Lab

Instructor: Nitin Chandrachoodan **nitin@ee.iitm.ac.in**

Moodle: **https://coursesnew.iitm.ac.in/**

Overview

- Purpose of the course
- Mode of operation
- Expectations

Practical

- Python
- Documentation

Purpose of the course

Assumptions

- You already know *some* programming - probably C, maybe others
- You already know something about EE

Goal

- How do we write programs to implement EE concepts?

EE Concepts?

- Kirchhoff's laws: V-I-R relations
    - Nodal analysis of simple circuits
- Combinational logic gates, netlists
    - Logic simulation, event driven simulation
- Electrostatics
    - Electric field under charge distribution
- Others from your favourite branch of EE

Other useful concepts

- Data analysis and fitting
    - Curve fitting, data modeling
- Monte Carlo techniques
    - Using randomness to solve (possibly) deterministic problems
- Optimization
    - Open-ended optimization problems and techniques

Mode of operation

- **Lecture timings:** Wednesdays 2:00 to 3:15 pm
- **Location:** CRC 102
- **Assignments:** On Moodle - **https://coursesnew.iitm.ac.in/**
- **Announcements:** Through Moodle announcements. **Check your email regularly**.

Assignments

- Weekly assignments distributed through Moodle
    - Some Python notebooks, some PDF problem statements
- Submission through Moodle
    - Code
    - Documentation

Assignment Evaluation

- Demo to TA
    - Demonstrate functionality
    - Answer questions - may include request for live modifications
- Documentation
    - Can be generated from Python notebook - but must be formatted properly
    - Marks for quality: content as well as aesthetics
- Possible bonus questions - will require significant work beyond problem statement
- Penalties for **late submissions**

You need to ensure your assignments are evaluated on time and marks uploaded to Moodle

Grading

- Major part of grade based on assignment submissions: TA evaluation
- Highest grades (A and S) require additional work
    - Example problems will be given as part of problem statement
    - Requires clear evidence of individual work
- Exams: to be decided
    - Most likely 1 or 2 time-limited programming assignments

Attendance

- 1 lecture per week: will NOT be recorded - you are expected to attend in person
    - Problem statement will be made available on Moodle
- IE lab reserved for 2 sessions per week
    - Primarily meant for demo sessions with TAs
    - Can also be used for doubt clearing, getting suggestions on coding, debugging help etc.
- Programs expected to be completed as *home work* - lab time used for debugging and/or demos

Assignment submissions will be taken as a metric of attendance. Late submissions will result in loss of attendance

Plagiarism

- **You are expected to write your own code**
  - Various checks *may* be applied - random, MOSS, other similarity checks
  - Getting away with copied code on occasion **does not mean** it is permitted!
- Instances of copied code will result in immediate loss of marks
  - We will not try to identify who copied from whom - both parties lose
- Extreme instances will be referred to Disciplinary Committee

Plagiarism - what?

- Discussion with friends - permitted, but you are expected to write your own code
    - Document the code the way you understand it
    - we will take size and complexity of the code into account
- Explanations: mandatory
    - if you cannot explain code you are supposed to have written, that is a red flag and will result in serious loss of marks
- StackOverflow, github etc
    - Provide clear citations of sources and explain why you did not write it yourself

Yes, we are aware

of ChatGPT.
And no, you are not allowed to use it.

Philosophy

> *You can take a horse to the water, but you cannot make it drink*

Expectations

- Overall course credit: 6
    - Implies 5 hours per week of work (6 lecture hour equivalents)
- 1 - 1.5 lecture hours per week in actual lecture
- ~10-15 minutes per week demo to TA (0.25 - 0.5 lecture hour)
- Remaining time: ~ 4 to 4.5 hours per week
    - Understanding the problem and framing the solution
    - Programming
    - Debugging
    - Documentation

Problem

- Not all of you have same skill levels at programming and debugging
- Problems formulated so that systematic effort will result in *good* grade
- But top grade requires significant extra effort

Python

- Programming language of choice for the course
    - Generic - not too opinionated on programming style
    - Good libraries and ecosystem - high performance through foreign functions
    - Excellent documentation and support system
- The bad
    - Performance
    - Environment management can be tricky
    - Can hide complexity that a system programmer should be aware of

References

- **https://docs.python.org/3/tutorial/**
- **https://cs50.harvard.edu/python/2022/** - generally good reputation
- w3schools, tutorialspoint etc. - use your judgement

Main Concepts

- Interpreted
- Dynamic typing
- Multi-paradigm
    - Imperative: most common starting point
    - Functional: programming as a composition of functions
    - Object-oriented: data knows what you can do with it

Interpreted Language

- No ahead-of-time (AOT) compilation. May do some Just-in-time (JIT) compilation
    - Finally need to reduce high level language to machine instructions.
    - Interpretation does this at run time
- Problems:
    - Speed: cannot do many optimizations
    - Errors: caught at run-time, not compile time
- Benefits:
    - Speed of development: just write and run; modification easy
        - **REPL** - read-evaluate-print-loop
    - ?

Types - C

C has static types: declared with code:

```c
int a;
float b;

a = 10;   // valid - checked at compile time
b = 10;   // valid - int treated as "sub-type" of float
a = 2.5; // automatic coercion
b = "hello"; // invalid - compile time error
```

Advantages:

- Catch many type related errors at compile time
- Optimizations possible: if you know something is int, use more efficient algorithms

Disadvantages:

- User has to specify all types ahead of time
- Type coercion useful for convenience but can cause misunderstanding (implicit conversions from float to int etc.)

Types - Python

Python variables are dynamically typed

```python
a = 1   # treated as int
a = 2.5 # no problem.  from now a will be treated as float
a = "hello"  # still no problem.  Now it will be treated as a string
```

Advantages:

- Easily write code without worrying about the type of an object. Development speed can increase

Disadvantages:

- Compiler cannot do optimizations without knowing data types
- Since type of a variable can change in future, JIT compilation also suffers

Imperative Paradigm

- Tell the computer **how** to solve the problem
- Each step has to be broken down
    - Fits with the idea of "algorithm" as a sequence of steps
- Programmer has full control, but also full responsibility

```
In [1]: def fact(n):
           prod = 1
    for i in range(1, n+1): # specify how to compute
        prod = prod * i
    return prod
print(fact(5))
```

```
120
```

Functional Paradigm

- More mathematical formulation of problem solving:
    - Composition of functions
    - Higher order functions: function-of-a-function
- Provides some good insights on how hardware can be modeled
- Allows a more "declarative" paradigm: you specify **what** needs to be done rather than how exactly to compute it

```
In [2]: def factf(n):
            if n <= 1:
           return 1
       else:
           return n * factf(n-1)   # What to compute - how is decided by languag
print(factf(5))
```

```
120
```

Object-oriented Paradigm

- "Real-world objects"
    - properties and restrictions on how they can be manipulated
- Encapsulation
- Inheritance
- Passing messages between objects

Module system

- Python modules like C libraries
    - Encapsulate functionality for use by others
- Packaging, versioning
- `import`
- Namespaces

One of the main reasons for Python's success: though it is a slow language, it can invoke functionality from highly optimized libraries through the module system

What do you need to know?

- Various data types - how and when to use them
    - Numeric, String, Lists, Dictionaries, Objects
    - Numeric arrays, matrices (special libraries like numpy)
- Basic control
    - Control flow: if, while, for
    - Functions, comprehensions
    - Classes, objects
- How to write from scratch
    - eg. Construct a matrix and solve linear equations
- How to use Modules
    - Use an appopriate module to get better performance

Documentation

- Self-documenting code:
    - Not just lots of comments - they must be meaningful and useful
- Description of problem solution:
    - Text and **Figures**
    - Code samples with suitable output where needed

Markup (Markdown)

- Include annotations in text to indicate how to format
- This is opposed to "presentation"
- Examples:
    - HTML
    - Markdown
    - LaTeX
- Use a compilation step to convert the output to a good format