Revised 4/2025

NOVA COLLEGE-WIDE COURSE CONTENT SUMMARY
ITP 250 – ADVANCED PYTHON PROGRAMMING (4 CR.)

Course Description

Object-oriented design and advanced programming concepts using Python through instruction and hands-on programming. Emphasizes Object-oriented design (OOD) Object Orient Programming (OOP) concepts, such as classes, inheritance, polymorphism, Object-oriented design patterns, and Unified Modeling Language (UML). Examines best practices, code reusability, and exploration of Python modules and advanced topics.
Lecture 4 hours. Total 4 hours per week.

General Course Purpose

This course provides instruction and hands-on programming requirements for a student to create, modify, test, and debug Python programs that contain object-oriented constructs and advanced Python components.

Course Prerequisites/Corequisites

Prerequisite: ITP 150.

Course Objectives

Upon completing the course, the student will be able to:

- Design, code, test, and debug Python programs in a hands-on approach using an Integrated Development Environment (IDE) to implement concepts covered in this course
- Use their foundational knowledge of Python, write code that:
  - Illustrates a full range of object-oriented constructs (OOP)
  - Illustrates more advanced features of the Python language
- Write code using the Python Standard Library
- Import and write code that uses various Python modules and/or third-party modules
  Perform OOD tasks using UML. Implement the design using OOP constructs and OO best practices

Major Topics to be Included

- Hands-on programming using a Python Integrated Development Environment (IDE)
- Continued programming and advanced use of various foundational constructs, including:
  - Python Standard Library and modules, importing
  - Variables and Python data types, including lists, tuples, and dictionaries
  - If statements and loops
  - Functions. Function and method calls
  - Code that opens, reads, and writes to files. Use of appropriate corresponding data structures.
- Object-oriented design and object-oriented programming, including:
  - OOD and use of UML for various OO concepts covered in this course.
  - OOP and best practices for various OO concepts covered in this course
  - Classes, attributes, and methods
  - Creation of objects and calling of methods
  - Inheritance, polymorphism, abstract methods / classes
  - Static vs. instance constructs
  - OO design patterns and best practices
- Recursion and advanced function / method code

- Best practices regarding code reusability, reduction of code maintenance, and testing
- Third-party modules
- Exploration of additional advanced topics using Python. May include additional Python modules, common frameworks, and/or advanced programming concepts.

Student Learning Outcomes

- Use an Integrated Development Environment (IDE) to write, update, and test Python code
    - Use a debugger program to walk-through the Python code
- Explain each of the following foundational constructs:
    - Explain and identify arguments and parameters
    - Explain the difference between modules, functions, and methods
    - Explain the difference between built-in functions and user-defined functions
- Write Python code that calls built-in functions, user-defined functions, and methods.
- Write Python code that has/uses each of the foundation constructs:
    - Data structures including lists, tuples, and dictionaries
    - If statements
    - While loops
    - For loops
    - Creation of user-defined functions
    - Calling functions passing arguments
- Write Python code that uses a main function
    - Write Python code that follows the best practice of calling main by checking the __main__ attribute
- Explain the difference between immutable and mutable data types. Identify Python built-in data types.
- Continue to work with text files
    - Write Python code that will:
        - Open, read, and close a file. Read the file contents into an appropriate data structure.
        - Open, write to, and close a file. Write the file contents from an appropriate data structure.
    - Explain the difference between the file modes of read, write, and append and the relationship of these with the file permission maintained by the operating system. Write code that utilizes these different file modes.
    - Understand the .txt file type. Also, understand the following text file types and their required structure. Match and use with a corresponding Python data type. Write code that uses (writes to or reads from) the following file types:
        - .csv
            - Use the csv Python module to program with this file type.
            - Use with a list of lists or other appropriate Python data type.
        - .json
            - Use with a Python dictionary.
            - Use the json Python module to program with this file type.
            - Use with string, int, float, list, and embedded dictionaries.
            - Be able to observe the contents of a json file and interpret the contents, their relationships, and the appropriate Python data type(s).
            - Given a problem statement, design the dataset and json contents.
- Advanced topics regarding functions
    - Write and call a function that uses parameter default values
    - Call a function using named parameters
    - Explain difference between pass-by-reference and pass-by-value and how Python uses pass- by-reference because all variables are objects
    - Explain difference between immutable and mutable types and its implications when passing parameters of these types.
        - Example: behavior when passing an int (immutable) vs. a list (mutable) to a function

        - Write and call a function that accepts a mutable type (such as a list), and illustrate the mutability of the list and the implication to the calling function
    - Write and code a function that returns multiple objects. Illustrate options for capturing return value and how it is wrapped in a tuple.
        - Write code that uses recursion.Examples could include math factorial, traversing the recursion inheritance hierarchy.
        - Optional – draw a stack trace to illustrate recursion
- Explain exceptions their uses, and their benefits.
    - Write code that catches an exception
    - Write code that throws a new exception
    - Write code that catches an exception using a base exception class, that serves as an illustration that the exception hierarchy uses inheritance.
- Explain various OO concepts, including classes, objects, encapsulation, and data hiding
    - Explain the concept of public and private attributes (instance variables) and their relationship with data hiding and encapsulation. Explain the limitations of implementing this in Python. Explain how this can be simulated in Python using the dunder naming convention and the name mangling that occurs when dunder is used.
    - Explain the difference between local variables, instance variables and class variables. Identify the scope differences of each. Write code that illustrates these.
    - Explain the benefit of set and get methods and their relationship to data hiding and encapsulation
    - Explain the difference between accessors and mutators.
    - Write code that uses each of these concepts through one or more user-defined classes:
        - Private attributes (instance variables) using the dunder naming convention
        - Methods
            - Public instance methods
                - Constructor
                - set methods
                - get methods
                - str method
                - repr method
            - Class methods
            - Write each of these methods and also write code that will call each of these methods.
            - Write code that will invoke the str method and also code that will invoke the repr method and understand the difference.
        - Class variables
        - Create an immutable class
    - Write code to define these classes and also write code that will create objects of each class and call methods on the objects
    - Write code that will create multiple objects and place them into a list or other data structure. Call methods on the objects in the list.
    - Given problem statement(s), design the classes, methods, and OO features that are needed.
- Explain the relationships involved with inheritance and its benefits. Explain abstract methods, abstract classes, and polymorphism.
    - Write code that uses each of these concepts through one or more user-defined classes:
        - Inheritance
        - Abstract methods
        - Abstract base class
        - Use of inheritance with implementation and calls within the str method
        - Use of super
    - Explain how inheritance is used by the exception hierarchy
    - Given problem statement(s), design the class hierarchy, methods, and OO features that are needed.
- Create one or more UML OOD class diagrams that illustrate the following:
    - Classes with attributes and method signatures

    - Inheritance
    - Abstract classes
    - OO patterns and/or other OO common constructs
- Write code that illustrates code reusability. Explain the benefits of code reuse and benefits of code maintenance considerations.
- Other Python language features
    - Write code using the pass statement
    - Call the type() built-in function to determine the type. May be called using the Python shell.
- Advanced Topics, choose one or more of the following items:
    - Write code using additional modules within the Python Standard Library, such as csv, json, abc, sqlite3, and any others.
    - Write code using third-party modules and/or common Python frameworks.
    - Write code using GUI application development using Tkinter or a similar library.
    - Write code performing CRUD database actions and integration with Python data structures and functions.
    - Explain the definition, purpose, and benefits of a RESTful API. Review code or write code that uses a third-party module that enables use of an API.
    - Advanced Python topics at the discretion of the instructor

Required Time Allocation per Topic

To standardize the core topics of ITP 250 so that a course is equivalent in content across campuses and formats, the following student contact hours per topic are required. Each syllabus should be created to adhere as closely as possible to these allocations. Of course, the topics cannot be followed sequentially. Many topics are taught best as an integrated whole, often revisiting the topic several times, each time at a higher level. The topics listed should comprise 60 contact hours of instruction for a 4-credit class excluding the final exam regardless of the format of instruction. The final exam time is not included in the timetable.

Topic	Hours	Percent
Review of Python IDE and basics	4	6.5%
Advanced use of functions and additional non-OO Python constructs	4	6.5%
Files, file types, and corresponding Python data structures	4	6.5%
Object-Oriented Basics - Classes, Attributes, Methods, and Objects	12	20%
Object-Oriented Advanced Topics – Inheritance and OO Patterns	16	27%
Advanced Topics - Exploration of Python modules and additional advanced Python topics. May include OO concepts.	16	27%
Testing to include quizzes, tests, and exams (not including final exam)	4	6.5%
Total	60	100%