Lecture 2.6

Data Structures with Python: Constructors & Methods

Agenda

- 1. What is a Constructor?
- 2. The __init__ Method
- 3. Default vs. Parameterized Constructors
- 4. Overview of Method Types
- 5. Instance Methods
- 6. Class Methods (@classmethod)
- 7. Static Methods (@staticmethod)
- 8. Summary & Best Practices

Constructors & The __init__ Method

- A constructor is a special method automatically called when a new instance of a class is created.
- Its primary purpose is to initialize the attributes (state) of the object.
- In Python, the constructor is strictly defined as ‘__init__()‘.
- It is a 'dunder' (double underscore) method, signifying its special role in the Python language.
- The first parameter is always ‘self‘, referencing the newly created object.

Default vs. Parameterized Constructors

- Default Constructor:
- - Takes no arguments other than ‘self‘.
- - Sets object attributes to fixed default values.
- Parameterized Constructor:
- - Takes arguments in addition to ‘self‘.
- - Allows dynamic assignment of attribute values at the time of object instantiation.
- - Provides flexibility when creating multiple distinct objects.

Code Example: Python Constructors in Action

- “‘python class Student: # Parameterized Constructor def __init__(self, name, enroll_no): self.name = name self.enroll_no = enroll_no
  # Object Instantiation triggers __init__ s1 = Student("Alice", "IT001") print(s1.name, s1.enroll_no) # Output: Alice IT001 “‘
- Notice how the arguments 'Alice' and 'IT001' map to 'name' and 'enroll_no'.

Types of Methods in Python: Overview

- Python provides three distinct ways to define methods within a class, based on their interaction with class data:
- 1. Instance Methods: Bound to the object. Access instance and class state.
- 2. Class Methods: Bound to the class. Access and modify only class state.
- 3. Static Methods: Unbound utility functions. Access neither instance nor class state.

Instance Methods

- The most common type of method in Python classes.
- Defined with ‘self‘ as the first mandatory parameter.
- Through ‘self‘, they can access instance attributes and other instance methods.
- Through ‘self.__class__‘, they can also access class-level attributes.
- Used for behaviors specific to individual objects.

Code Example: Instance Methods

- “‘python class Rectangle: def __init__(self, length, width): self.length = length self.width = width
  def calculate_area(self): # Instance Method return self.length * self.width
  rect = Rectangle(10, 5) print(”Area:”, rect.calculate_area()) # Output: 50 “‘

Class Methods & The @classmethod Decorator

- Class methods operate on the class itself, not on object instances.
- Defined using the ‘@classmethod‘ decorator above the function signature.
- Takes ‘cls‘ as the first mandatory parameter, representing the class.
- Used for modifying class state that applies across all instances, or providing alternative constructors.

Code Example: Class Methods

- “‘python class Employee: company_name = ”TechCorp” # Class variable @classmethod def change_company(cls, new_name): cls.company_name = new_name Employee.change_company(”InnovateInc”) print(Employee.company_name) # Output: InnovateInc “‘

Static Methods & The @staticmethod Decorator

- Defined using ‘@staticmethod‘. They take neither ‘self‘ nor ‘cls‘.
- They behave like plain functions but reside in the class namespace for logical organization.
- “‘python class MathOperations: @staticmethod def is_even(number): return number % 2 == 0 print(MathOperations.is_even(10)) # Output: True “‘
- Use them when a method does not need to access class or instance properties.

Summary and Method Comparison

- Constructors (‘__init__‘): Crucial for safely initializing object state.
- Instance Methods: Require ‘self‘. Access & modify object data.
- Class Methods: Require ‘@classmethod‘ & ‘cls‘. Access & modify class data.
- Static Methods: Require ‘@staticmethod‘. Independent functions within class scope.
- Choosing the right method type ensures robust and clear object-oriented designs.