ADVANCED PYTHON

Joanne Wang

KnightHacks | Jan. 29 2025

JOANNE WANG

- Third-year Computer Science Major
- Machine Learning and VR Researcher
- Uses Python a lot

FOLLOW ALONG!

https://axiomatic-cry-7f6.notion.site/Advanced-Python-18a8c2c956c780fe9a06ff884644ff6f

CONTENTS

DATA STRUCTURES

DATA STRUCTURES

- Contents:
  - Sets
  - Heaps
- Good to Know
  - Lists
  - Dictionaries
  - Linked Lists

SETS

- Unordered collection of unique elements
- Key Operations:
  - **add**
  - **remove**
  - **in**
- Use Cases:
  - Checking for uniqueness (e.g., filtering duplicates)
  - Fast membership checks
  - Mathematical set operations (union, intersection, etc.)
- Time Complexity:
  - Add, remove, and in: O(1) on average

HEAPQ

- A heap is a binary tree with special properties:
  - Min-heap: The root node contains the smallest element
  - Max-heap: The root node contains the largest element
- The heapq module provides an efficient way to implement a priority queue using a min-heap
  - Use negative values with heapq for max-heap

HEAPQ

- Core Functions:
  - **heapq.heappush(heap, item)** – Adds an element to the heap, maintaining the heap property
  - **heapq.heappop(heap)** – Removes and returns the smallest element from the heap
  - **heapq.heappushpop(heap, item)** – Pushes an item onto the heap and pops the smallest element
  - **heapq.heapify(iterable)** – Transforms a list into a heap in-place
  - **heapq.nlargest(n, iterable)** – Returns the n largest elements from the iterable
  - **heapq.nsmallest(n, iterable)** – Returns the n smallest elements from the iterable

COLLECTIONS

COLLECTIONS

- Built-in Python Module
- Contents:
  - Counter
  - Deque
  - Defaultdict
  - Namedtuple

COUNTER

- Subclass of built-in dict to quickly count hashable objects.
  - Count occurrences of elements
  - Perform mathematical operations (+, –)
  - Retrieve most common elements
- Use Cases:
  - Word frequency counting
  - Finding the most frequent elements in a collection
  - Analyzing input data for patterns (e.g., in strings or lists)
- Time Complexity:
  - Count: O(n), where n is the length of the iterable

DEQUE

- Deque (double-ended queue) is a generalization of stacks and queues. Allows appending and popping from both ends with O(1) time complexity
- Key Operations:
  - **append/appendleft**
  - **pop/popleft**
- Use Cases:
  - Queues, stacks, and sliding window problems
  - Efficiently managing tasks in a task scheduler or buffer
- Time Complexity:
  - Append and pop from either end: O(1)
  - Insert or remove elements from the middle: O(n)

DEFAULTDICT

- Provides default value for nonexistent keys (Avoids KeyError)
  - Specify default value type (e.g., int, list, etc.)
  - Automatically initializes missing keys with the default value
- Use Cases:
  - Grouping elements into lists or sets (e.g., creating an adjacency lists)
  - Counting elements (like Counter but with custom default value)
  - Efficiently handling missing keys in dictionaries
- Time Complexity:
  - O(1)
  - Resizing the dictionary: O(n)

NAMEDTUPLE

- Factory function for creating tuple subclasses with named fields. Essentially a lightweight object with named attributes.
  - Access fields using dot notation
  - Use for lightweight objects that don’t require full-fledged classes
- Use Cases:
  - Storing simple objects where immutability is desired
  - Creating tuples with named fields for readability
- Time Complexity:
  - Creation: O(1)
  - Accessing fields: O(1)

COMPREHENSIONS

COMPREHENSIONS

- Concise syntax for creating lists, sets, dictionaries, etc.
  - List comprehension:
    - [expression for item in iterable if condition]
  - Set comprehension:
    - {expression for item in iterable if condition}
  - Dictionary comprehension:
    - {key: value for item in iterable if condition}

LAMBDA

LAMBDA

- A concise, anonymous function defined using the lambda keyword
  - lambda arguments: expression
  - Typically used for small, throwaway functions

MAP & FILTER & SORT

MAP & FILTER & SORT

- map(): Applies a function to all items in an iterable
  - map(function, iterable)
  - Returns an iterator of the results
- filter(): Filters items from an iterable based on a condition
  - filter(function, iterable)
  - Returns an iterator with items for which the function returns True
- sorted(): Sorts an iterable based on a given key function
  - sorted(iterable, key=function, reverse=False)
  - Returns a new list (does not modify the original)

ZIP

ZIP

- zip():
  - Combines multiple iterables element-wise into tuples
  - Can be used to parallelize iterations

OBJECT-ORIENTED PROGRAMMING

OOP

- What is Object-Oriented Programming?
  - Programming paradigm based on objects and classes
- Key Concepts:
  - Abstraction
  - Polymorphism
  - Inheritance
  - Encapsulation

DUNDERS

DUNDERS

- Magic methods (also known as dunder methods) are special methods in Python that begin and end with double underscores (__method__)
- They allow you to define how objects of a class behave in certain situations, such as arithmetic operations, comparisons, string representations, and more

DUNDERS

- Commonly Used Magic Methods:
  - __init__ – Constructor method for initializing an object
  - __str__ – Defines the string representation of an object (for print())
  - __repr__ – Defines the official string representation (for debugging)
  - __add__ – Defines behavior for the + operator
  - __len__ – Defines behavior for the len() function
  - __eq__ – Defines behavior for the equality operator ==
  - __lt__ – Defines behavior for the less-than operator <

ITERATORS & GENERATORS

ITERATORS

- An iterator is any Python object that implements the iterator protocol, which consists of the following two methods:
  - **__iter__()**: Returns the iterator object itself. This is required for creating an iterable object.
  - **__next__()**: Returns the next item in the sequence. If there are no more items, it raises a StopIteration exception.

GENERATORS

- A generator is a function that returns an iterator using the yield keyword instead of return
- The generator function pauses its execution at the yield statement and resumes from where it left off when the next item is requested
- Benefits of Generators:
  - **Memory Efficient:** Generate items lazily without loading everything into memory at once
  - Simpler Syntax: The generator function automatically implements the iterator protocol
- Lazy evaluation: Don’t need the entire sequence, yield items one by one

ENUM

ENUM

- What are Enums?
  - A set of symbolic names bound to unique, constant values
- Advantages of Using Enums:
  - Readability: Enums give meaningful names to values instead of using raw integers or strings
  - Safety: Prevent invalid values from being used
  - Maintainability: If a value changes, you only need to update it in one place (the Enum definition)

DECORATORS

DECORATORS

- A decorator is a function that takes another function as input and extends or modifies its behavior without changing the function's code.
- Why Use Decorators?
  - Code reusability across multiple functions
  - Avoid redundant code
- Common Use Cases
  - Logging: Log function calls automatically
  - Authorization: Check user permissions before running a function
  - Caching: Store results to avoid redundant computations
  - Performance: Measure execution time or memory usage

QUESTIONS?