Class 2 - Python Programming Review I

Guiding Genetic Algorithms with Language Models for Combinatorial Optimization

MISE Research Program - July 2025

Pedro Paredes

Demo: Looking at assignment solutions

Object Oriented Programming

Objects and Classes

Variables in Python are all *Objects* (so an *Object* is the most generic data type)

In practice they are collections of **attributes and methods**

A **class** is a blueprint or template that defines the structure and behavior of objects

```python
class Person:
    name = ""

    def greet(self):
        print("Hello!")

p = Person()
p.name = "Pedro"
print(p.name)
p.greet()
```

```python
class Circle:
    radius = 1.0
    color = "Blue"

c1 = Circle()
print(3.14 * c1.radius ** 2)
```

Why Classes?

- **Encapsulation**: Classes let us collect attributes and methods into a single unit
- **Code Reusability**: Once a class is defined, we can use it to create multiple objects
- **Abstraction**: A way to represent real-world or abstract concepts in code
- **Inheritance**: Creating new classes based on existing ones
- **Polymorphism**: Ability of objects of different classes to respond to the same method in different ways *(we are not going to need this)

Self and init

- *self* is used to access attributes and methods of a class in python
- *self* is a convention and not a python keyword
- Class functions must have an extra first parameter in the method definition
- *__init__*(also called constructor) function of a class is invoked when we create an object variable or an instance of the class

A longer Person example

```
1  class Person:
2      def __init__(self, name):
3          self.name = name
4
5      def greet(self):
6          print("Hello! My name is", self.name)
7
8  p = Person("Pedro")
9  p.greet()
```

More examples

```python
class Car:
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year

    def display_info(self):
        print("Car:", self.make, self.model, "(", self.year, ")")

    def start_engine(self):
        print("Engine started!")

# Creating an object of the Car class
my_car = Car("Toyota", "Corolla", 2022)

my_car.display_info()   # Output: Car: Toyota Corolla (2022)
my_car.start_engine()   # Output: Engine started!
```

Inheritance

- Inheritance allows derived classes (child classes) to inherit attributes and methods from a base class (parent class)
- Every class inherits from a built-in basic class called ‘object’
- Inheritance is achieved by specifying the base class name in parentheses after the derived class name during class declaration
- If a method is defined in both the base class and the derived class, the method in the derived class overrides the one in the base class

More examples

```python
class Animal:
    def __init__(self, name):
        self.name = name

    def speak(self):
        print("The animal makes a sound.")

class Dog(Animal):
    def speak(self):
        print("Woof!")

class Cat(Animal):
    def speak(self):
        print("Meow!")

dog = Dog("Buddy")
cat = Cat("Whiskers")

dog.speak()  # Output: Woof!
cat.speak()  # Output: Meow!
```

More examples

```python
class Animal:
    def __init__(self, name):
        self.name = name

    def eat(self):
        print("Animal", self.name, "is eating...")

class Bird(Animal):
    def fly(self):
        print("Bird", self.name, "is flying...")

canary = Bird("Tweety")

canary.eat()  # Output: Animal is eating...
canary.fly()  # Output: Bird is flying...
```

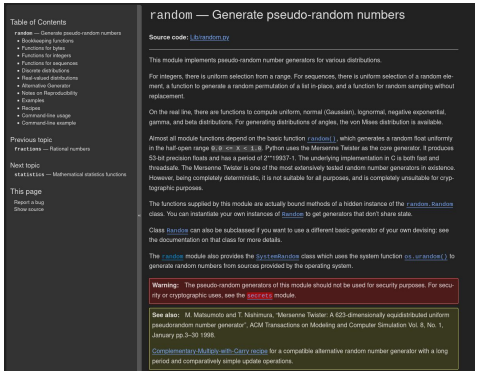
APIs and Libraries

What's a Python Library?

A ***library*** (also known as ***API***) is a collection of modules (Python files) that contain functions and classes

We can ***import*** a library into our program to be able to use classes/functions of the library

There is a lot of documentation online on what libraries provide, e.g. https://docs.python.org/3/library/random.html



Anatomy of a library usage

import *library*

this imports the library into our program, so we can use all of its methods and classes

library.method()

we can use the library methods if we write its name first followed by “dot” method

The random Library

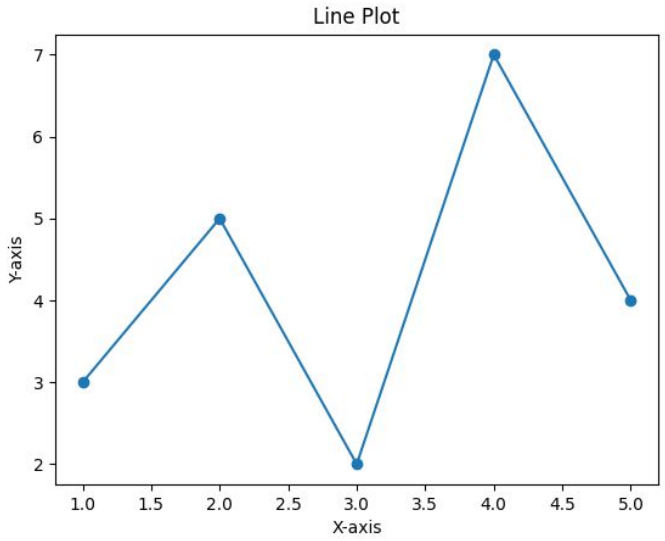
We can use the *random* library to generate (pseudo-)random numbers and randomize things

```
1   import random
2
3   print(random.random())
4   print(random.randint(1, 6))
5   pets = ["cat", "dog", "hamster"]
6   print(random.choice(pets))
```

```
 8   cards = list(range(1, 53))
 9   random.shuffle(cards)           # in-place permutation
10   hand = random.sample(cards, 5)  # 5 distinct cards
11   print(hand)
```

Matplotlib

- Matplotlib is a widely used data visualization **library** in Python
- It provides a comprehensive set of tools for creating various types of plots, charts, and graphs



Installing Packages with pip

Python comes with a few very useful libraries, but sometimes we want more, e.g. matplotlib

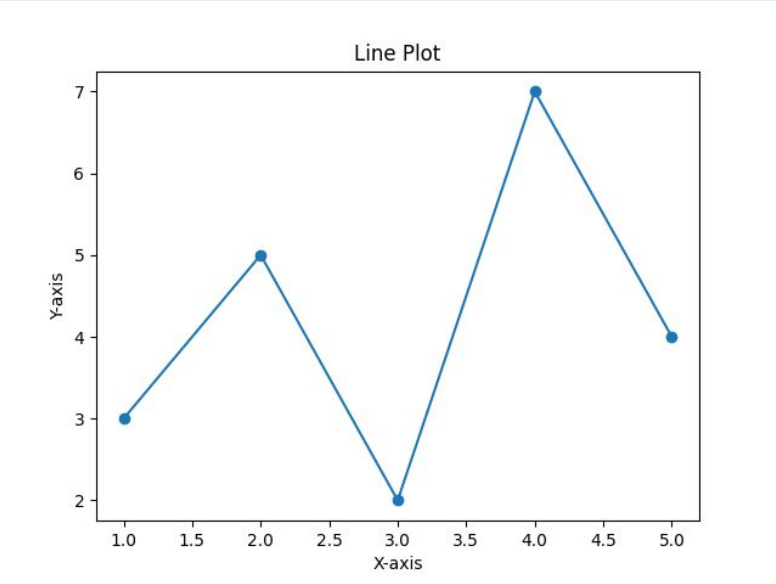
The ***Python Package Installer*** (known as ***pip***) is a command-line tool to download and install libraries

Open a terminal (on VSCode go to Terminal > New Terminal), type the following and press enter

```
python -m pip install matplotlib
```

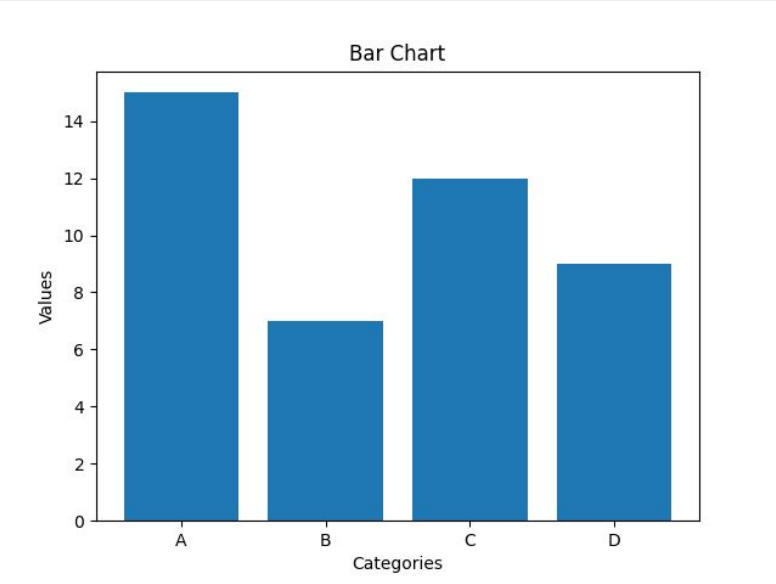
This will install matplotlib on your computer

Example: Line Plot



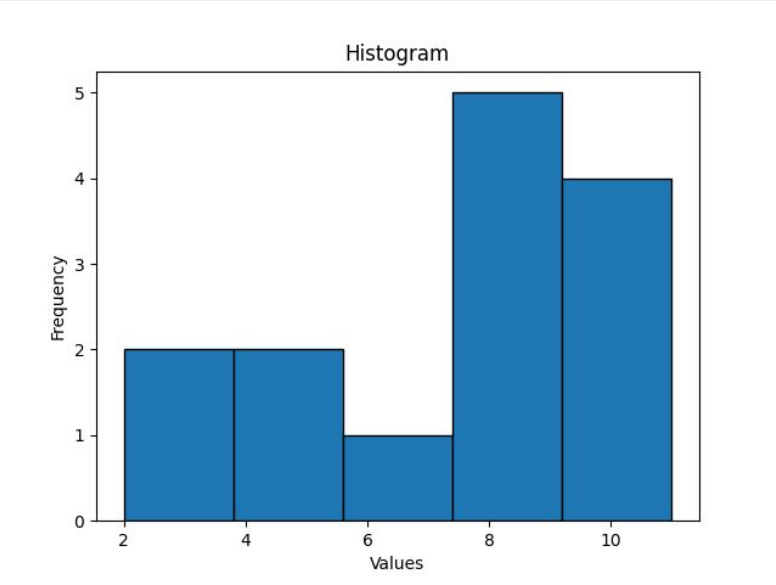

```python
import matplotlib.pyplot as plt

# Data
x = [1, 2, 3, 4, 5]
y = [3, 5, 2, 7, 4]

# Create a line plot
plt.plot(x, y, marker='o')

# Customize labels and title
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Line Plot')

# Display the plot
plt.show()
```

Example: Bar Chart




```python
import matplotlib.pyplot as plt

# Data
categories = ['A', 'B', 'C', 'D']
values = [15, 7, 12, 9]

# Create a bar chart
plt.bar(categories, values)

# Customize labels and title
plt.xlabel('Categories')
plt.ylabel('Values')
plt.title('Bar Chart')

# Display the plot
plt.show()
```

Example: Bar Chart




```python
import matplotlib.pyplot as plt

# Data
values = [2, 3, 5, 5, 7, 8, 8, 8, 9, 9, 10, 11, 11, 11]

# Create a histogram
plt.hist(values, bins=5, edgecolor='black')

# Customize labels and title
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.title('Histogram')

# Display the plot
plt.show()
```

OpenAI

Connect to the OpenAI API and prompt is LLM. It needs to be installed (information here: https://platform.openai.com/docs/libraries)

```python
from openai import OpenAI
client = OpenAI()

completion = client.chat.completions.create(
    model="gpt-4.1-nano",
    max_completion_tokens=30,
    messages=[
        {
            "role": "user",
            "content": "Write one sentence about Python."
        }
    ]
)

print(completion.choices[0].message.content)
```

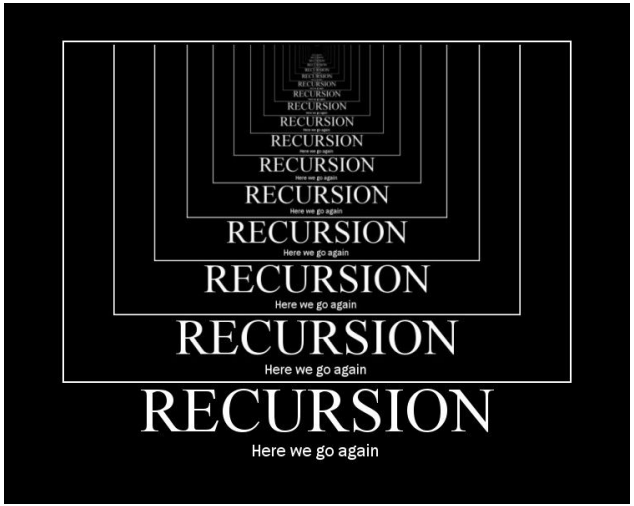
Recursion Recap

Recursion: What is it?

A problem solving approach where we break a problem into smaller versions of the same problem.

Technically, we can think of recursion as being a **function that calls itself**.

However, in reality, it turns out to be a powerful way to solve problems.



Simple Recursive Functions: Example 1

Let us consider the problem of **adding all the numbers in a list**.

If we implement a iterative (i.e. nonrecursive) solution for the problem, it would look like this:

```
def listSum(nums):
    numsum = 0
    for num in nums:
        numsum += num
    return numsum
```

How would a recursive version of this function look like?

Simple Recursive Functions: Example 1

A recursive solution for the problem would look like:

```
def recListSum(nums):
    if len(nums) == 0:
        return 0
    return nums[0] + recListSum(nums[1:])
```

List slicing: all elements of *nums* except the first one

Recursion: What is it?

We often divide a recursive function in two parts:

- A **base case**: returns a result for a known value;
- A **recursive case**: computes a result calling the same function for a different value.

In other words, with recursion, we solve a problem by assuming it is already solved :)

Recursion: Code example

A template for simple recursive functions can be achieved as follows:

```
def recursiveTemplate(value):
    if baseCase == True:
        return knownValue
    else:
        return recursiveTemplate(modify(value))
```

Recursion: Code example

A template for simple recursive functions can be achieved as follows:

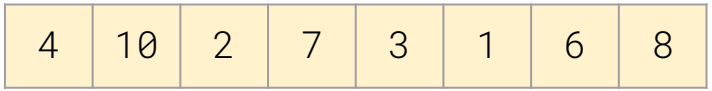
Base case

```
def recListSum(nums):
    if len(nums) == 0:
        return 0
    return nums[0] + recListSum(nums[1:])
```

Recursive case

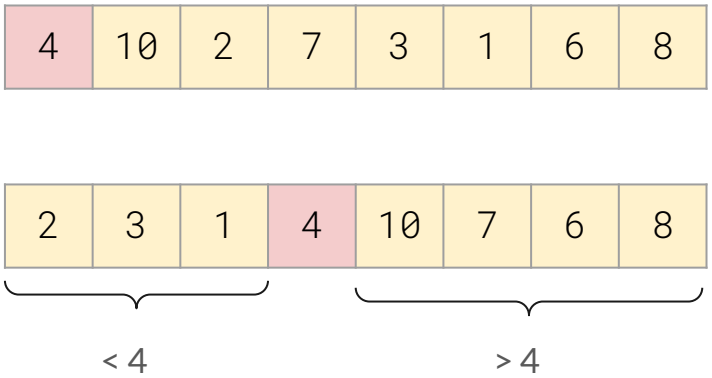
Sorting

Suppose you have a list of integers and you want to sort it



Let's try to use recursion by breaking the problem into smaller subproblems

Pick a number and move numbers less than it to the left, and greater than to the right



This process is known as ***partition***

Sorting - Code

```python
def partition(nums):
    pivot = nums[0]
    left = []
    middle = []
    right = []
    for num in nums:
        if num < pivot:
            left.append(num)
        elif num > pivot:
            right.append(num)
        elif num == pivot:
            middle.append(num)
    return (left, middle, right)

def recursive_sort(nums):
    if len(nums) <= 1:
        return nums
    (left, middle, right) = partition(nums)
    return recursive_sort(left) + middle + recursive_sort(right)

print(recursive_sort([4, 10, 2, 7, 3, 1, 6, 8]))
# prints [1, 2, 3, 4, 6, 7, 8, 10]
```

This algorithm is known as ***quicksort***

What's next?

Next class tomorrow

Another assignment sheet will be added to the website shortly

Class 3: Python Programming Review II