CSC340/AI320 - Artificial Intelligence
Fall 2022

Lab 3

INSTRUCTOR: DR. DIAA SALAMA ABDEL MONEIM
ENG. NOHA EL MASRY
ENG. NOUR ELHUDA ASHRAF

I.Functions and Generators:

These are all somewhat advanced tools that, depending on your job description, you may not encounter on a regular basis. Because of their roles in some domains.

Before designing your functions, you should know the following terms and guidelines:

- Cohesion: how you should divide tasks into purposeful functions.
- Coupling: how your functions should communicate with each other

➢ Function Design Concepts:

1. Each function should have a single, unified purpose.

When designed well, each of your functions should do one thing, something you can summarize in a simple declarative sentence.

2. Each function should be relatively small.

This naturally follows from the preceding goal, but if your functions start spanning multiple pages on your display, it is probably time to split them. Especially given that Python code is so concise to begin with, a long or deeply nested function is often a symptom of design problems. Keep it simple and keep it short.

3. Make a function independent of things outside of it.

Arguments (input to function) and return statements (output from function) are often the best ways to isolate external dependencies to a small number of well-known places in your code.

4. Use global variables when only necessary.

Global variables are usually a poor way for functions to communicate. They can create dependencies and timing issues that make programs difficult to debug, change, and reuse.

5. Do not change mutable arguments unless the caller expects it.

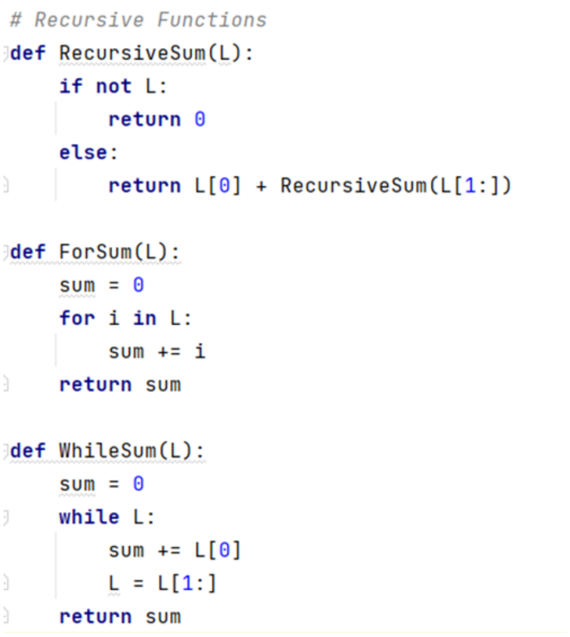
Functions can change parts of passed-in mutable objects, but (as with global variables) this creates a tight coupling between the caller and callee, which can make a function too specific and brittle.

6. Avoid changing variables in another module file directly.

Changing variables across file boundaries sets up a coupling between modules similar to how global variables couple functions, the modules become difficult to understand and reuse. Use accessor functions whenever possible, instead of direct assignment statements.

II. Recursive Functions:

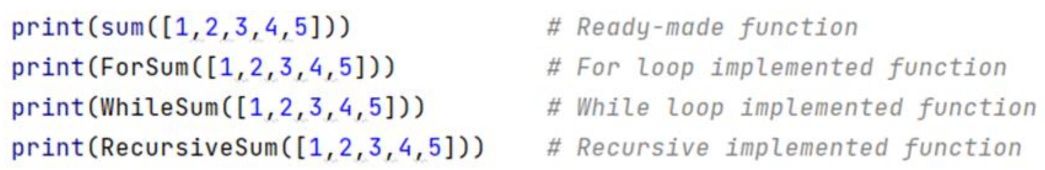
It is relatively rare to see recursive functions in Python, since procedural statements include simple loop structures. But it is still a useful and powerful technique to learn.



```python
# Recursive Functions
def RecursiveSum(L):
    if not L:
        return 0
    else:
        return L[0] + RecursiveSum(L[1:])

def ForSum(L):
    sum = 0
    for i in L:
        sum += i
    return sum

def WhileSum(L):
    sum = 0
    while L:
        sum += L[0]
        L = L[1:]
    return sum
```



```python
print(sum([1,2,3,4,5]))             # Ready-made function
print(ForSum([1,2,3,4,5]))          # For loop implemented function
print(WhileSum([1,2,3,4,5]))        # While loop implemented function
print(RecursiveSum([1,2,3,4,5]))    # Recursive implemented function
```

III. Modules and Packages:

A module is a Python file containing Python statements and definitions.

Similar codes are gathered in a module. This helps in modularizing the code and makes it much easier to deal with. Not only that, a module grants reusability. With a module, we do not need to write the same code again for a new project that we take up. To import all statements defined in modules we use the wild character (*).

What happens when importing a specific module?

A package is like a directory holding sub-packages and modules.

While we can create our own packages, we can also use one from the Python Package Index (PyPI). You cannot import a function using the dot operator(.) For that, you must type this:

from functools import reduce

A package must have the file __init__.py, even if you leave it empty.

So, what is the difference between module and package?

We will learn more about Modules such as Pandas, NumPy and Packages such as Matplotlib, SciPy and Scikit-Learn through the course when needed

IV. Classes and OOP:

You have to know that OOP is entirely optional in Python.

You can do your work in a procedural manner as in functional programming. Though, structural programming is a useful tool to implement long-term product development.

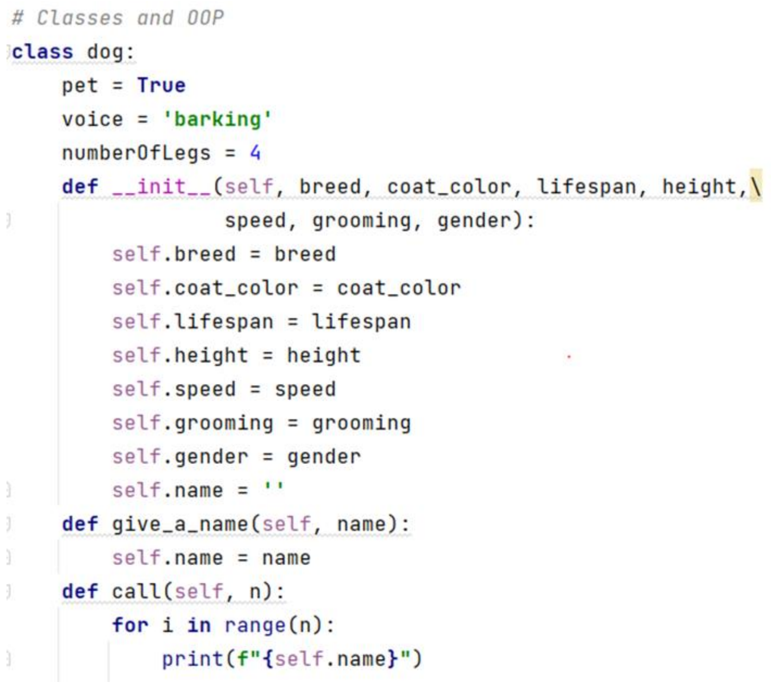
How to define a class?

As you priorly know from other courses, classes are the blueprint/factory of user-defined objects. Each class is associated with some properties called attributes and some functions called methods. Attributes are categorized into two types: class attributes and instance attributes.

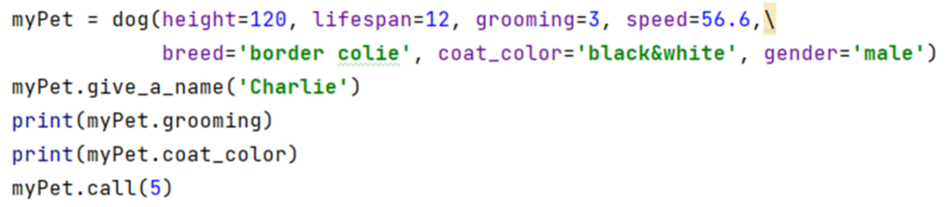
Class Attributes are defined directly beneath class definition (indented by 4 spaces). It has the same exact value for all instances of the class.

While instance properties (attributes) must be defined inside __init__().

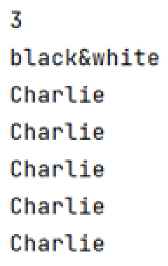
Every time an object/instance is created, __init__() sets the initial state of the object by assigning the values of the object's properties. That is, __init__() initializes each new instance of the class. Remember class constructor!



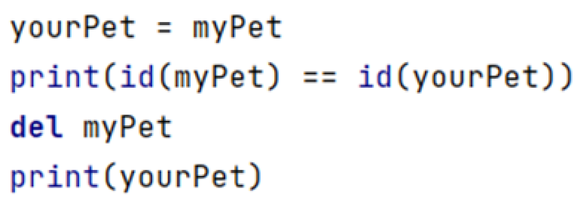
```python
# Classes and OOP
class dog:
    pet = True
    voice = 'barking'
    numberOfLegs = 4
    def __init__(self, breed, coat_color, lifespan, height,\
                 speed, grooming, gender):
        self.breed = breed
        self.coat_color = coat_color
        self.lifespan = lifespan
        self.height = height
        self.speed = speed
        self.grooming = grooming
        self.gender = gender
        self.name = ''
    def give_a_name(self, name):
        self.name = name
    def call(self, n):
        for i in range(n):
            print(f"{self.name}")
```



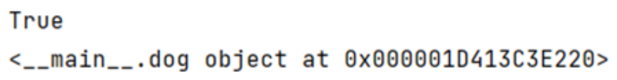
```python
myPet = dog(height=120, lifespan=12, grooming=3, speed=56.6,\
            breed='border colie', coat_color='black&white', gender='male')
myPet.give_a_name('Charlie')
print(myPet.grooming)
print(myPet.coat_color)
myPet.call(5)
```



```
3
black&white
Charlie
Charlie
Charlie
Charlie
Charlie
```



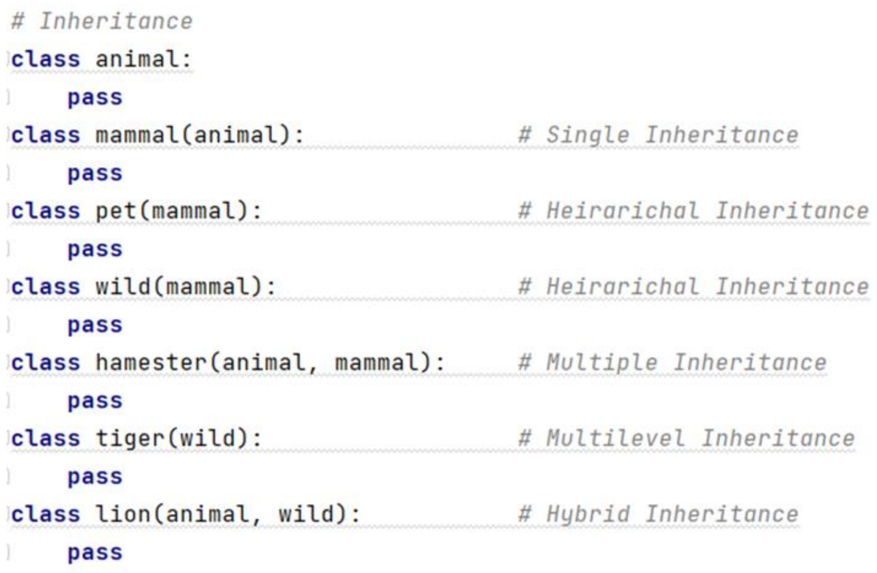
```python
yourPet = myPet
print(id(myPet) == id(yourPet))
del myPet
print(yourPet)
```



```
True
<__main__.dog object at 0x000001D413C3E220>
```

Inheritance

To make a class inherit from another, we apply the name of the base class in parentheses to the derived class' definition.



```
# Inheritance
class animal:
    pass
class mammal(animal):                # Single Inheritance
    pass
class pet(mammal):                   # Heirarichal Inheritance
    pass
class wild(mammal):                  # Heirarichal Inheritance
    pass
class hamester(animal, mammal):      # Multiple Inheritance
    pass
class tiger(wild):                   # Multilevel Inheritance
    pass
class lion(animal, wild):            # Hybrid Inheritance
    pass
```

Does Python support super() function? If so, what is its purpose?

Does Python support method override? Does it also support method overload?

Exercises:

1. Write the definition of a Point class. Objects from this class should have the following members:

o x and y coordinates.

o a method show to display the coordinates of the point

o a method move to change these coordinates in the direction x or/and y.

o a method dist that computes the distance between 2 points.

Also write a main function to test your implementation.

2. Write a Python program to create a Vehicle class with max_speed and mileage instance attributes.

-Create a Bus class that inherits from the Vehicle class. Give the capacity argument of Bus.seating_capacity() a default value of 50.

-Create a Bus child class that inherits from the Vehicle class. The default fare charge of any vehicle is seating capacity * 100. If Vehicle is Bus instance, we need to add an extra 10% on full fare as a maintenance charge. So total fare for bus instance will become the final amount = total fare + 10% of the total fare.

3. Create a Python class called BankAccount which represents a bank account, having as attributes: accountNumber (numeric type), name (name of the account owner as string type), balance.

-Create a constructor with parameters: accountNumber, name, balance.

-Create a Deposit() method which manages the deposit actions.

-Create a Withdrawal() method which manages withdrawals actions.

-Create an bankFees() method to apply the bank fees with a percentage of 5% of the balance account.

-Create a display() method to display account details.

-Give the complete code for the BankAccount class

4. Write a Rectangle class in Python language, allowing you to build a rectangle with length and width attributes.

Create a Perimeter() method to calculate the perimeter of the rectangle and Area() method to calculate the area of the rectangle.

Create a method display() that display the length, width, perimeter and area of an object created using an instantiation on rectangle class.

Create a Parallelepipede child class inheriting from the Rectangle class and with a height attribute and another Volume () method to calculate the volume of the Parallelepiped.