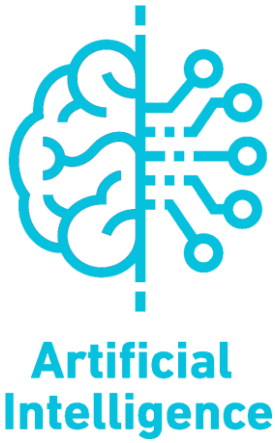
AI320 - Artificial Intelligence

Spring Semester 2022

Instructor: Dr. Diaa Salama Abdel Moneim
Eng. Haytham Metawie
Eng. Mostafa Badr
Eng. Nour Elhuda Ashraf



MARCH 8, 2022

Lab 3 – Introduction to Python - Part 3

Topic	Duration
Introduction to Python - Part 3	2 hours

Document Revision History

Revision Number	File Name	Date
1	Lab 3 – Introduction to Python - Part 3	8/3/2022

IV. 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.

➢ Recursive Functions

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

```
# 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
```

```
15
15
15
15
```

```
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
```

Code Alternatives

```
# Code Alternatives
def sum1(L):
    return 0 if not L else L[0] + sum1(L[1:])          # Use ternary expression
def sum2(L):
    return L[0] if len(L) == 1 else L[0] + sum2(L[1:])  # Assume one case of the inputs
def sum3(L):
    first, *rest = L
    return first if not rest else first + sum3(rest)    # Use 3.X ext seq assignment
```

➢ Filter & Reduce Functions

```
# Filter & Reduce Functions
from functools import reduce
L1 = list(range(-5,6))
L2 = list(filter((lambda x: x > 0), range(-6,6)))
L3 = reduce((lambda x, y: x + y), L1)

print(L1)
print(L2)
print(L3)
```

```
[-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5]
[1, 2, 3, 4, 5]
0
```

V. 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 make 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 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.

VI. Classes and OOP:

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

You can do your work in procedural manner as in functional programming. Though, structural programming is a useful tool to implement a 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 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!

```
# 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}")
```

Default Constructor: What if we did not define a constructor?
Non-Parametrized Constructor: __init(self)___
Parametrized Constructor: as depicted in the above figure.
N.B. No constructor overloading in Python.

➢ Accessing class members (attributes and methods)

```
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
```

N.B We can use assignment operator to construct an object as follows:

```
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?

VII. Errors & Exception Handling:

try/except (Error Handling)

Catch and recover from exceptions raised by Python, or by you.

```
try
    #do something
except #Error or Exception
    #print an error message
```

try/finally (Termination Action)

Perform cleanup actions, whether exceptions occur or not.

```
try
    #do something
except #Error or Exception
    #print an error message
finally
    #termination action; executed every time.
```

raise (Unusual Control Flow)

Trigger an exception manually in your code.

```
if #condition:
    raise #Exception
```

Assert (Special Case Handling)

Conditionally trigger an exception in your code.

```
assert #condition, #Error Text
```

with/as (Termination Action)

Implement context managers in Python 2.6, 3.0, and later (optional in 2.5).

```
with open('file_path', 'w') as file:
    file.write('hello world !')
```

Exercises:

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

   - x and y coordinates.
   - a method show to display the coordinates of the point
   - a method move to change these coordinates in the direction x or/and y.
   - a method dist that computes the distance between 2 points.

   Also write a main function to test your implementation.

2. **A software system for university management has:**

   - Any of students or lecturers have attributes such as name, address, date of birth, email address and methods to access and set their attributes.
   - Additionally, student has ID, GPA. Student's cumulative GPA is calculated by (Total Points Earned)/(Total Credit Hours Attempted) . While lecturer has academic rank, salary and teaching load (number of teaching hours calculated via summing up his courses credit hours). Additionally, a bonus salary for more than 8 hours teaching load and is calculated by (extra teaching hours)/8*0.6*original salary.
   - Each lecturer object maintains a list of the courses he/she teaches. A lecturer may teach more than one course, but a course is taught by a teacher solely.
   - Course object has attributes such as course code, course title, course level and credit hours. Each course object maintains a list of the enrolled students and the lecturer who has been assigned to teach the course.
   - The course object has a behavior that allows enroll, unenroll student as well as assigning a lecturer. It also has methods such as getting the currently enrolled students and the current lecturer.

   Implement the above description exploiting the OOP concepts. Write a main function to test your implementation.

Thank you