LECTURE 4 FUNCTIONAL PROGRAMMING, AND OOP

Functional Programming Tools

- Python can support many different programming paradigms including functional programming.
- Lambda function within Python.
  - A lambda function is a small, anonymous function defined using the **lambda** keyword.
  - It can take any number of arguments and has only one expression, which is evaluated and returned.
  - Syntax: lambda argumensts: expression
  - Characteristics:
    - A lambda function does not have a name (but can be assigned to a variable)
    - A lambda function only has a single expression. It is used for small function
    - A lambda function is often used in functional programming tools.

```
>>> add = lambda x, y: x + y
>>>
>>> def add(x, y):
>>>     return x+y
```

Lambda function

- Lambda function Characteristics:
  - A lambda function does not have a name (but can be assigned to a variable)
  - A lambda function only has a single expression. It is used for small functions
  - A lambda function is often used in functional programming tools.

```
>>> def f(x):
... return x**2
...
>>> print (f(8))
64
>>> g = lambda x: x**2
>>> print (g(8))
```

```
>>> g = lambda x, y: x + y
>>> print(g(10, 20))
```

Python conditional expression

- In C++, we have condition expression. Example: a = a > b ? a : b;
- Python conditional expression syntax:
    - value_if_true if condition else value_if_false
    - Example: a = a if a > b else b
- Can be used in a lambda function to support conditions

Functional Programming Tools

- Filter
  - filter(*function*, *sequence*) filters items from sequence for which function(*item*) is true.
  - Returns a string or tuple if sequence is one of those types, otherwise result is a list.

```
def even(x):
    if x % 2 == 0:
        return True
    else:
        return False

print(list(filter(even, range(0,30))))
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28]
```

- Exercise: rewrite the code in the right side with one line Lambda function

Functional Programming Tools

- Map
  - map(*function, sequence*) applies function to each item in sequence and returns the results as a list.
  - Multiple arguments can be provided if the function supports it.

```
>>> print(list(map(lambda x: x*3, [1,2,3,4]))
[3,6,9,12]
```

```
def expo(x, y):
  return x**y

print(list(map(expo, range(1,5), range(1,5))))

[1, 4, 27, 256]
```

Functional Programming Tools

- Reduce
  - Defined in functools module
  - reduce(*function, sequence*) returns a single value computed as the result of performing *function* on the first two items, then on the result with the next item, etc.
  - There’s an optional third argument which is the starting value.

```
import functools
def fact(x, y):
  return x+y
print(functools.reduce(fact, [10, 30, 11]))

51
```

Functional Programming Tools

- Small user defined function to be applied to the whole array – lambda function

```
>>> print(list(map(lambda x: x**2, range(0,11))))
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Exercise

-
  ```
  sales = [
      {"product": "Laptop", "quantity": 4, “unit_price": 800},
      {"product": "Phone", "quantity": 10, “unit_price": 500},
      {"product": "Tablet", "quantity": 3, “unit_price": 300},
     {"product": "Monitor", "quantity": 6, “unit_price": 150}
  ]
  ```

Write code to compute total sale

Object-Oriented Programming in Python

- Python is a multi-paradigm language and, as such, supports OOP as well as a variety of other paradigms.
- If you are familiar with OOP in C++, for example, it should be very easy for you to pick up the ideas behind Python’s class structures.

Class Definition

- Classes are defined using the class keyword with a very familiar structure:

```
class ClassName :
    statement1
    …
    statementN
```

- There is no notion of a header file to include so we don't need to break up the creation of a class into declaration and definition. We just declare and use it!

Class Object

- Class example and its use.

```
class MyClass:
""""A simple example class docstring"""
i = 12345  # this is a class variable (static variable)
def f(self):
return MyClass.i
```

- Create a new instance of MyClass

```
 x = MyClass()
    print(x.f)
```

- See `lect4/class1.py`

Constructor

- Define the special method __init__() which is automatically invoked for new instances (initializer).

```
class MyClass:
    """A simple example class""“
    i = 12345
    def __init__(self):
        print "I just created a MyClass object!"
    def f(self):
        return 'hello world'
```

Constructor

- Variables defined outside of __init__() are class variables
- Variables defined inside __init__() are elements of the object

```
class MyClass:
"""A simple example class""“
i = 12345 # this is a class variable
def __init__(self):
        self.j = 10  # an element of the object
                      self.k = 20 # an element of the object
def f(self):
return self.k
```

- See lect4/class2.py

Data Attributes

- Like local variables in Python, there is no need for a data attribute to be declared before use.
- A variable created in a class is a static variable.
- To make instance variables, they need to be prefixed with “self”. This is especially evident with mutable attributes.
- There are also some built-in functions we can use to accomplish the same tasks.

Built-in attributes

- Besides the class and instance attributes, every class has access to the following:
  - __dict__: dictionary containing the object’s namespace.
  - __doc__: class documentation string or None if undefined.
  - __name__: class name.
  - __module__: module name in which the class is defined. This attribute is "__main__" in interactive mode.
  - __bases__: a possibly empty tuple containing the base classes, in the order of their occurrence in the base class list.
- See lect4/class2_n.py

Private Variables

- No mechanism to distinguish private and public variables in the language
- Achieve the same goal by naming convention
  - If an attribute is prefixed with a single underscore (e.g. _name), then it should be treated as private. Basically, using it should be considered bad form as it is an implementation detail.
  - To avoid complications that arise from overriding attributes, Python does perform name mangling. Any attribute prefixed with two underscores (e.g. __name) and no more than one trailing underscore is automatically replaced with _classname__name.

Inheritence

- The basic format of a derived class is as follows:

```
class DerivedClassName(BaseClassName):
        statement1
        ...
        statementN
```

- In the case of BaseClass being defined elsewhere, you can use module_name.BaseClassName.
- See lect4/class3.py