**PYTHON WEEK 4 INHERITANCE**

In object-oriented programming, inheritance is the mechanism of basing an object or class upon another object (prototype-based inheritance) or class (class-based inheritance), retaining similar implementation

**OOP in Python: A Brief Introduction**

Object oriented programming (OOP) is a way of designing programs using **objects** that have **properties** and **behaviors** related to a problem domain.

**Private Variable**

“Private” instance variables that cannot be accessed except from inside an object don’t exist in Python. However, there is a convention that is followed by most Python code: a name prefixed with an underscore (e.g. \_spam) should be treated as a non-public part of the API (whether it is a function, a method or a data member). It should be considered an implementation detail and subject to change without notice.

Since there is a valid use-case for class-private members (namely to avoid name clashes of names with names defined by subclasses), there is limited support for such a mechanism, called name mangling. Any identifier of the form \_\_spam (at least two leading underscores, at most one trailing underscore) is textually replaced with \_classname\_\_spam, where classname is the current class name with leading underscore(s) stripped. This mangling is done without regard to the syntactic position of the identifier, as long as it occurs within the definition of a class.

Name mangling is helpful for letting subclasses override methods without breaking intraclass method calls. For example:

class Mapping:

def \_\_init\_\_(self, iterable):

        self.items\_list = []

        self.\_\_update(iterable)

def update(self, iterable):

     for item in iterable:

            self.items\_list.append(item)

\_\_update = update   # private copy of original update() method

class MappingSubclass(Mapping):

def update(self, keys, values):

     # provides new signature for update()

     # but does not break \_\_init\_\_()

     for item in zip(keys, values):

            self.items\_list.append(item)

The above example would work even if MappingSubclass were to introduce a \_\_update identifier since it is replaced with \_Mapping\_\_update in the Mapping class and \_MappingSubclass\_\_update in the MappingSubclass class respectively.

Note that the mangling rules are designed mostly to avoid accidents; it still is possible to access or modify a variable that is considered private. This can even be useful in special circumstances, such as in the debugger.

Notice that code passed to exec() or eval() does not consider the classname of the invoking class to be the current class; this is similar to the effect of the global statement, the effect of which is likewise restricted to code that is byte-compiled together. The same restriction applies to getattr(), setattr() and delattr(), as well as when referencing \_\_dict\_\_ directly.