Python tutorial

Vasyl Vovk, Consultant, Globallogic

Confidential

Agenda

1. args and kwargs;
2. Decorators;
3. Classes.

Confidential

GlobalLogic

args and kwargs

Confidential

args and kwargs

```
def demo(a, b, *args, **kwargs):
    print(a)
    print(b)

    print(args)

    print(kwargs)


demo(1, 10)

demo(11, 20, 111,112,113)

demo(21, 30, 211,220, key1=221, key2=333)
```

Confidential

GlobalLogic

Decorators

Confidential

Decorators

In Python, functions are the first class objects, which means that:

- Functions are objects; they can be referenced to, passed to a variable and returned from other functions as well.
- Functions can be defined inside another function and can also be passed as argument to another function.

Decorators are very powerful and useful tool in Python since it allows programmers to modify the behavior of function or class. Decorators allow us to wrap another function in order to extend the behavior of wrapped function, without permanently modifying it.

In Decorators, functions are taken as the argument into another function and then called inside the wrapper function.

Decorators

Decorators in python is syntax sugar and don't do any kind of "magic". For instance this code:

```
1   @decorator
2   def my_function():
3       pass
```

Is equals to:

```
1   def my_function():
2       pass
3
4   my_function = decorator(my_function)
```

Decorators

We can easily implement caching for function using decorators:

```
1   def memoize(func):
2       cache = {}
3
4       def wrapper(*args, **kwargs):
5           key = args, tuple(kwargs.items())
6
7           if key not in cache:
8               cache[key] = func(*args, **kwargs)
9
10          return cache[key]
11
12      return wrapper
```

Confidential

Decorators

Now decorator can be applied to function:

```
13    @memoize
14    def fibonacci(n):
15        x, y = 0, 1
16
17        for i in range(n):
18            x, y = y, x + y
19
20        return x
```

When function will be called second time, no calculation will be performed:

```
>>> fibonacci(10)
>>> fibonacci(10) # on second call, value from cache will be used
```

Confidential

Decorators

You can apply several decorators for one functions. Let's create decorator to measure time of function execution:

```
1    import time
2
3    def timed(func):
4
5        def wrapper(*args, **kwargs):
6            start = time.time()
7            ret = func(*args, **kwargs)
8            print(time.time() - start)
9            return ret
10
11       return wrapper
```

Decorators

```
@timed
@memoize
def fibonacci(n):
    x, y = 0, 1

    for i in range(n):
        x, y = y, x + y

    return x
>>> fibonacci(10)
2.5170528888702393
>>> fibonacci(10)
0
```

GlobalLogic

Classes

Confidential

Scopes and Namespaces

```
def scope_test():
    def do_local():
        spam = "local spam"
    def do_nonlocal():
        nonlocal spam
        spam = "nonlocal spam"
    def do_global():
        global spam
        spam = "global spam"
    spam = "test spam"
    do_local()
    print("After local assignment:", spam)
    do_nonlocal()
    print("After nonlocal assignment:", spam)
    do_global()
    print("After global assignment:", spam)

scope_test()
print("In global scope:", spam)
```

Classes

the definitions for the data format and available procedures for a given type or class of object; may also contain data and procedures (known as class methods) themselves, i.e. classes contains the data members and member functions

Confidential

Classes

```
class File:
    object_counter = 0
    def __init__(self, file_name, method):
        self.file_obj = open(file_name, method)
    def __enter__(self):
        return self.file_obj
    def __exit__(self, type, value, traceback):
        self.file_obj.close()
    def run(self):
        pass

f = File("text.txt", "w")
```

Class variable

Instance method

Instance variable

Class instance

Class instance

In object-oriented programming (OOP), an instance is a concrete occurrence of any object, existing usually during the runtime of a computer program. Formally, "instance" is synonymous with "object" as they are each a particular value (realization), and these may be called an instance object; "instance" emphasizes the distinct identity of the object. The creation of an instance is called instantiation.

Confidential

Static data

```
def get_no_of_instances(cls_obj):
    return cls_obj.no_inst

class Kls(object):
    no_inst = 0

    def __init__(self, data):
        Kls.no_inst = Kls.no_inst + 1

ik1 = Kls()
ik2 = Kls()

print(get_no_of_instances(Kls))
```

Confidential

Static/Class method

```
class File:
    def __init__(self, data):
        self.data = data

    def printd(self):
        print(self.data)

    @staticmethod
    def smethod(*args):
        print('Static:', args)

    @classmethod
    def cmethod(*args):
        print('Class:', args)
```

Class. Empty

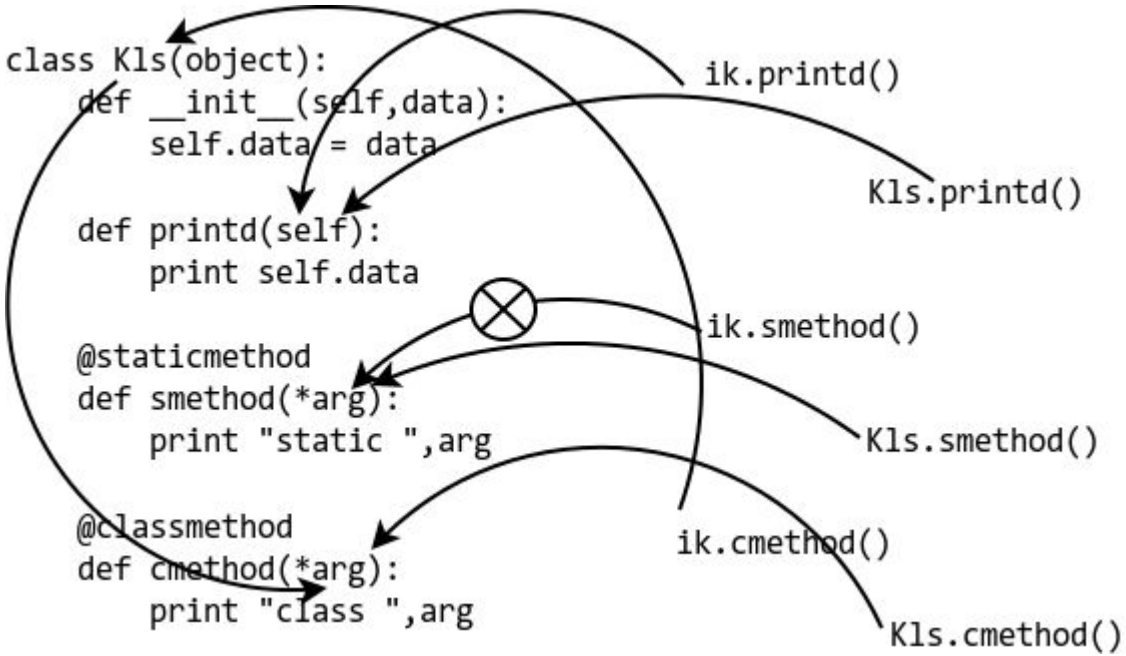
```
class Student:
     pass

 john = Student()  # Create an empty employee record

 # Fill the fields of the record
 john.name = 'John Doe'
 john.dept = 'computer lab'
 john.scholarship = “4000$”
```

Confidential

Things explained



Inheritance

```
class DerivedClassName(BaseClassName):
      <statement-1>
      .
      .
      .
      <statement-N>
```

Inheritance

The name BaseClassName must be defined in a scope containing the derived class definition. In place of a base class name, other arbitrary expressions are also allowed. This can be useful, for example, when the base class is defined in another module

Confidential

Inheritance

Exercise: create class hierarchy for Animals

Confidential

Inheritance. Built-in function

Python has two built-in functions that work with inheritance:

- Use isinstance() to check an instance's type: isinstance(obj, int) will be True only if obj.__class__ is int or some class derived from int
- Use issubclass() to check class inheritance: issubclass(bool, int) is True since bool is a subclass of int. However, issubclass(unicode, str) is False since unicode is not a subclass of str (they only share a common ancestor, basestring)

Multiple inheritance

```
class DerivedClassName(Base1, Base2, Base3):
      <statement-1>
      .
      .
      .
      <statement-N>
```

Confidential

MRO

https://makina-corpus.com/blog/metier/2014/python-tutorial-understanding-python-mro-class-search-path

Confidential

Private Variables and Class-local References

```
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)
```

Confidential

Composition

is a way to combine simple objects or data types into more complex ones

Confidential

Aggregation

Aggregation is a type of composition. It Differs from ordinary composition in that it does not imply ownership.

In composition, when the owning object is destroyed, so are the contained objects. In aggregation, this is not necessarily true.

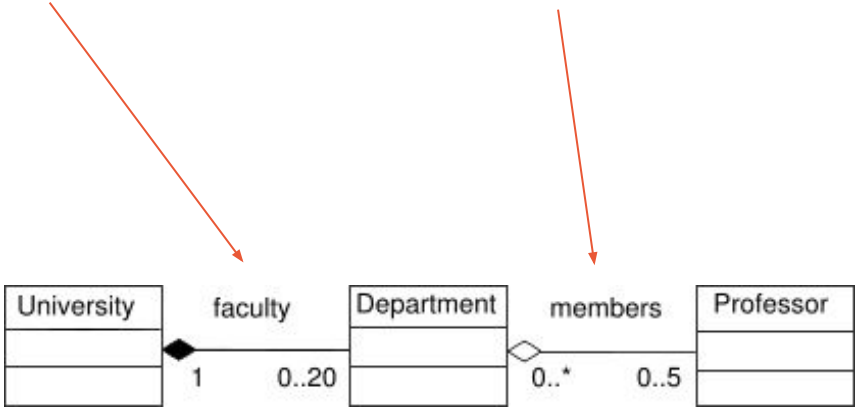
Confidential

Composition and Aggregation examples

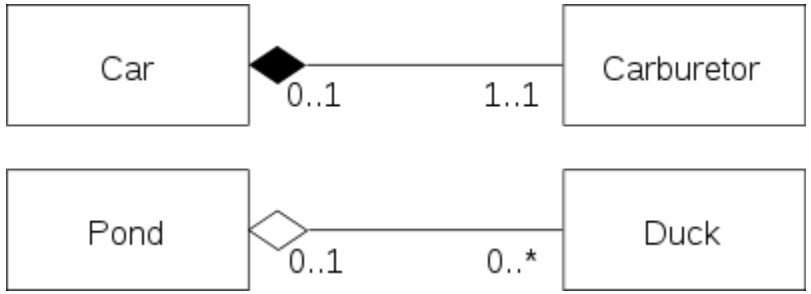
For example, a university owns various departments (e.g., chemistry), and each department has a number of professors. If the university closes, the departments will no longer exist, but the professors in those departments will continue to exist. Therefore, a University can be seen as a composition of departments, whereas departments have an aggregation of professors. In addition, a Professor could work in more than one department, but a department could not be part of more than one university.

Confidential

Composition and Aggregation examples



Composition

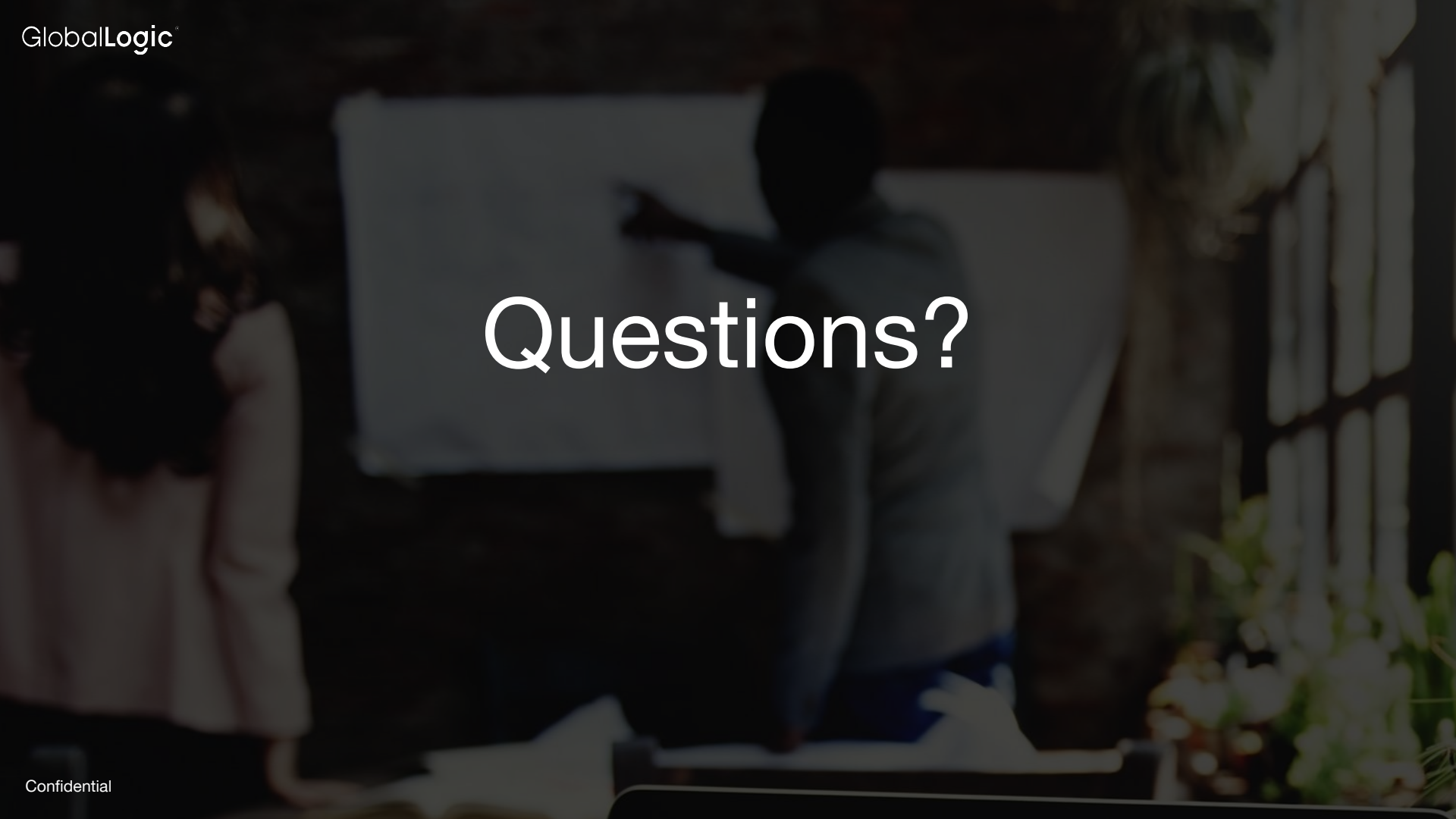


Aggregation

Magic Methods

Object.__method__

https://www.python-course.eu/python3_magic_methods.php



Questions?

Confidential