VE477
Introduction to Algorithms

Lab 2

Manuel — UM-JI (Fall 2018)

Goals of the lab

- Efficient C implementations
- Object oriented programming in Python
- More advanced Python topics

1 C programming

Using the C standard library write efficient implementations of

- The `Union-Find` data structure.
- Kruskal's algorithm.
- Prim's algorithm for solving the Minimum Spanning Tree problem.

Consider the complexity of Kruskal and Prim algorithms and then compare how do they do perform in practice.

Hint: run the implementations on various types of graphs (size, sparse, dense...).

2 More advanced programming in Python

Modules

A module is a "Python library", that is a file where some functions, variables, or classes are defined. The main difference with regular C or C++ libraries is that a module can also include some runnable function.

Using modules

```
print(sqrt(9))
print(math.cos(0))
# import a specific function into the current name space
from math import sqrt
print(sqrt(9))
# import a whole module
import math
print(dir(math))
print(math.cos(0))
```

Modules can be easily installed using the package manager on Linux. At times it might however happen that modules are not packaged. In such a case, or on other operating systems, it is recommended to use the Python package installer `pip`.

Writing modules

The following module is composed of the function `leonardo` and an if statement. The function returns all the Leonardo numbers less than *n*. In itself this function is enough to define the module which can later be imported. The last three lines however allows the function to be directly called from the command line and arguments passed to it.

Leonardo numbers (leo.py)

```
#!/usr/bin/python3

def leonardo(n):
    a, b = 1, 1
    seq=[a]
    while b < n:
        seq.append(b)
        a,b = b, a + b + 1
    return seq

if __name__ == "__main__":
    import sys
    print(leonardo(int(sys.argv[1])))
```

Running from the command line.

```
sh $ python3 leo.py 1000
```

Using as a module.

```
import leo
leo.leonardo(1000)
print(dir(leo))
```

Object oriented programming in Python

In Python the type of an object is represented by the class used to build the object.

Types

```
a=3; b=5.0
type(a); type(b)
```

Classes

More than one class can be implemented in a .py file. New keywords starting with "@" can be used to simplify and clarify the class implementation.

Base room class (room.py)

```python
#!/usr/bin/python3

class Room:
# attributes common to all instances
    windows = 2
    color = 'white'

# constructor
    def __init__(self, name, size, status):
        self.name = name
        self.size = size
        self.status = status

# methods common to all instances
    @classmethod
    def paint(cls, color):
        cls.color = color
        print(cls.color)

# methods called without reference to the class or the instance
    @staticmethod
    def knock():
        print("Anybody here?")

# transforms a method into a read-only attribute (get)
    @property
    def size(self):
        return self.name + ' is ' + str(self._size) + ' square meters'

# allows to set the previous read-only attribute
# without it, it would be impossible to set the size attribute
    @size.setter
    def size(self, size):
        self._size = size

# allows to delete the property, which can be set again later
    @size.deleter
    def size(self):
        del self._size

# methods
    def open(self):
        self.status = 'open'

    def lock(self):
        self.status = 'locked'
```

Example usage of the `Room` class.

```
from room import Room
b=Room('bedroom',10,'open')
b.size; b.size=200; b.size
b.status; b.lock(); b.status
b.knock()
l=Room('living room',20,'open')
# color is a class attribute
b.color; l.color; Room.color='red'; b.color; l.color
# color can be adjusted for each instance
b.color='green'; b.color; l.color
```

Inheritance

In Python inheritance works as usual.

Panic room class (panicroom.py)

```
#!/usr/bin/python3

from room import Room
from random import randint

class PanicRoom(Room):

    solidity = 0

    def ZombyAttack(self):
        strength = randint(0,100)
        if strength > self.solidity:
            print(self.name + ' is compromise, run!')
        else:
            print(self.name + ' is safe, stay in here!')
```

Example usage of the `PanicRoom` class.

```
from panicroom import PanicRoom
p=PanicRoom('secret hidden room',8,'locked')
p.size
p.solidity=50; p.ZombyAttack()
p.open()
```

Polymorphism

Polymorphism is extremely simple to realise in Python.

Panic room class (panicroom_poly.py)

```python
#!/usr/bin/python3

from room import Room
from random import randint

class PanicRoom(Room):

    solidity = 0

    def ZombyAttack(self):
        strength = randint(0,100)
        if strength > self.solidity:
            print(self.name + ' is compromise, run!')
        else:
            print(self.name + ' is safe, stay in here!')

# redefinition of open()
    def open(self):
        ans=input('Are you sure this is a good idea? (y/n) ')
        if ans == 'y':
            self.status = 'open'
```

The Python approach to polymorphism is much different than the one of C++.

```python
def echo(a):
  return a
echo(5)
echo('test')
echo(['one', 'two', 'three'])
```

From a C++ perspective the `echo` function has not been defined for all those data types so this should not be working. However the approach taken in Python is to assume the defined function is universal and does the same thing regardless of the data type.

In Python instead of covering all the cases that might arise, the problem is delegated to the data, expecting it to perform the correct operation. For instance when adding two numbers there is no need to define the function for all the possible combination of input data type (e.g. int + float, int + int, float + complex...).

This is well illustrated by the implementation of the `+` operator: when `a+b` is called, `a.__add__(b)` is executed.

```
a=3; b=4; a.__add__(b)
a=3.0; b=4; a.__add__(b)
a=3; b=4.0; a.__add__(b)
```

In particular this explains why functions do not need to specify their input and output data types.

Error handling

When coding in Python it is common directly access attributes or methods without first checking their existence. This is consistent with the polymorphism approach described in the previous section.
In case something goes wrong as exception is raise. This can be easily and cleanly handle as follows.

```
def length(a):
    try:
        len(a)
    except TypeError:
        print("len not supported by this data type")
# run if everything went well
    else:
        print(len(a))
# always run
    finally:
        print("clean up resources here")
# strings and lists have a length
length('abcdefg')
length([1,2,3,4])
# integers do not have a length
length(1)
```

Raised exception can be of various types such as `ttributeError`, `TypeError`, `NameError` and `IndexError`.

More Python

Explain how to use:

- The `with` statement;
- Decorators
- Iterators
- Generators

When running python3 from the command line, the `help()` function can be used to access a description of functions and keywords. More information can be found in the official Python 3 documentation.

The webpage 15 Essential Python Interview Questions considers interesting points to test your knowledge and understanding of Python.