Advanced Programming

Lesson 2:

Regular Expressions (10.5)

and

Classes (chapter 9)

Connect to:

network_lab

Password:

network3b

Head to:

10.10.6.37

Inheritance (2)

- Last lecture there were questions about how to call a parent class' method.
- In the simple example we just overrode the original tostring() method, without calling the original.
- We can call the parent class' method by using the super method:

```
class MastersStudent(Student):
    def tostring(self):
        description = super(MastersStudent,self).tostring()
        description = description + ' (M)'
        return description
```

We put the name of the current class here

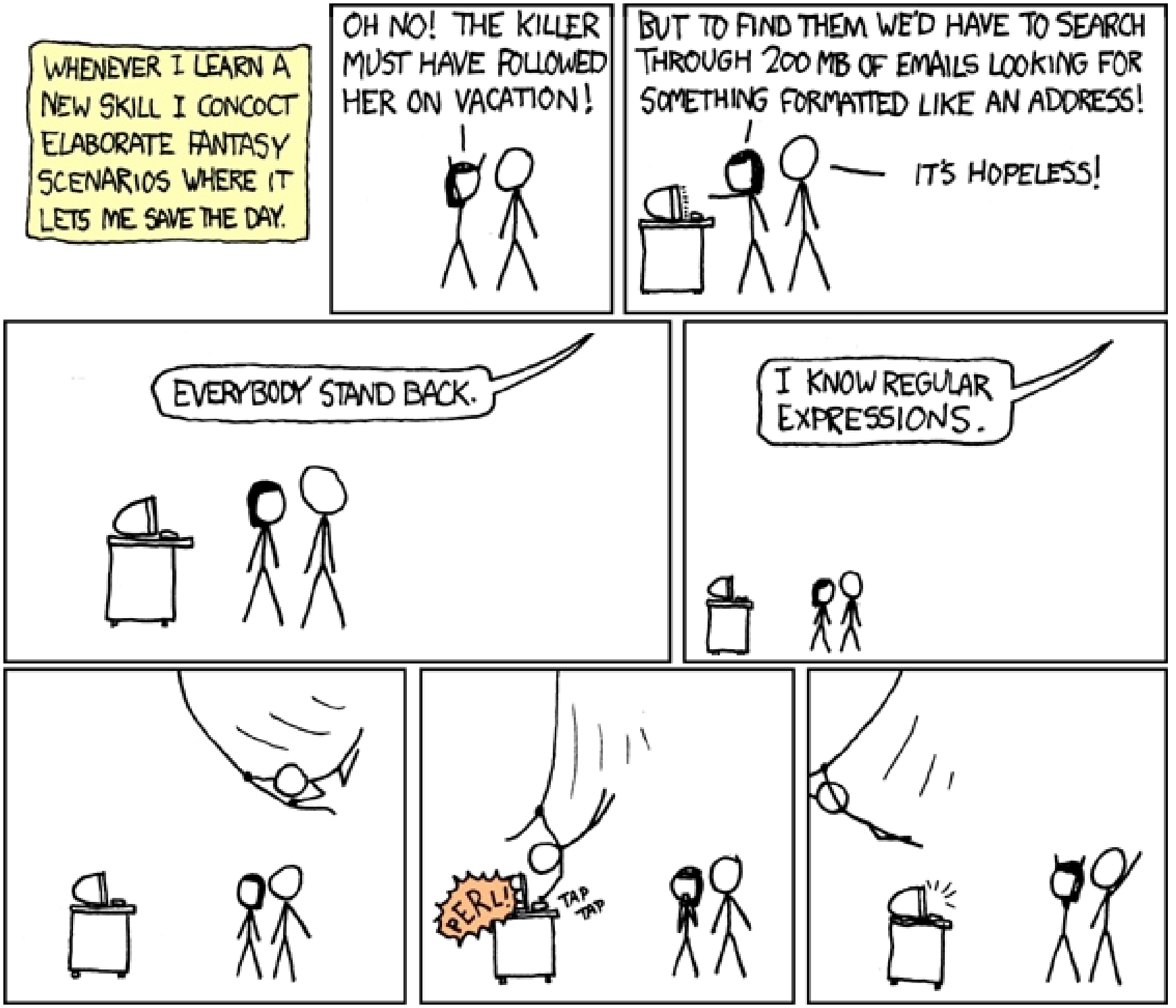
Inheritance (2)

- To make this work though we need to modify the parent class slightly, to descend from the **`object`** class:

```
class Student(object):
    def __init__(self,firstname,lastname,regno):
        self.firstname = firstname
        self.lastname = ... etc etc ...
             ...     ...
```

Regular Expressions

Regular expressions let you search and replace text.


WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.
OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!
BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!
IT'S HOPELESS!
EVERYBODY STAND BACK.
I KNOW REGULAR EXPRESSIONS.
PERL!
TAP TAP

xkcd

Regular Expressions

- Regular expressions are useful to process strings. You can **search** or **alter text** with them.
- First, a simple example:
  - To use them in python we need to import the regular expressions library:

```
import re
```

  - Example regexs:

```
print re.search('all','bAll')
re.findall('al*','balance allowed')
```

    - `['al', 'a', 'all']`

Regular Expressions

Regular expression

String being searched

```
re.findall('al*','balance allowed')
```

- `['al', 'a', 'all']`

Regular Expressions: Metacharacters: `.*^$+?[](){}|\`

`tre*`

The `*` is an example of a **metacharacter**.

The `*` (asterisk or star) matches when the preceding character occurs **0 or more times**,

For example, **`tre*`** will find:
`tree` (e is found 2 times)
`tread` (e is found 1 time)
`trough` (e is found 0 times).

Regular Expressions: Metacharacters: `.*^$+?[](){}|\`

`tre+`

The `+` matches when the preceding character occurs **1 or more times** (notice this is slightly different from the * which matches zero or more times).

For example, **`tre+`** will find:

`tree` (e is found 2 times)

`tread` (e is found 1 time)

~~`trough` (e is found 0 times).~~

This is not found as with a + it needs to appear at least once

Regular Expressions

- Work through the regexp pdf:
  - First, [character class] [^complement]
  - Special sequences
    - \d digits
    - \s spaces
    - \w alphanumeric
    - . Anything.
  - Repetitions a* a+ a?

Regex Golf

```
import regexgolf

print regexgolf.puzzlewords(1)['matchwords']

print regexgolf.puzzlewords(1)['rejectwords']

regexgolf.verifypuzzle('regex_here',1)
```

- The aim is to match the 'matchwords' and not match the reject words:

```
matchwords:['buffoon', 'foody', 'foolish', 'fool', 'catfoot', 'afoot',
'footlights', 'footmen', 'seafood']

rejectwords:['crinkle', 'palace', 'wildfowl', 'critical',
'spontaneous', 'forget', 'info']
```

Error: should match but did not: fool, footmen, seafood, catfoot, footlights, buffoon, foody, foolish, afoot

Classes etc...

- Look through chapter 9 to get an idea about classes...
- Download the *students.py* file
- Modify the classes in the following ways:
  - Add a **new attribute** to the Student class, regyear (year of registration).
  - **Create a new class Lecturer,** and **extend** the definition of the Course class so that lecturers can be added.

End of week 2's lecture