Basics of programming

Ralf Lämmel
Software Languages Team, University of Koblenz-Landau

© 2014, Ralf Lämmel, rlaemmel@acm.org

The notion of *program*

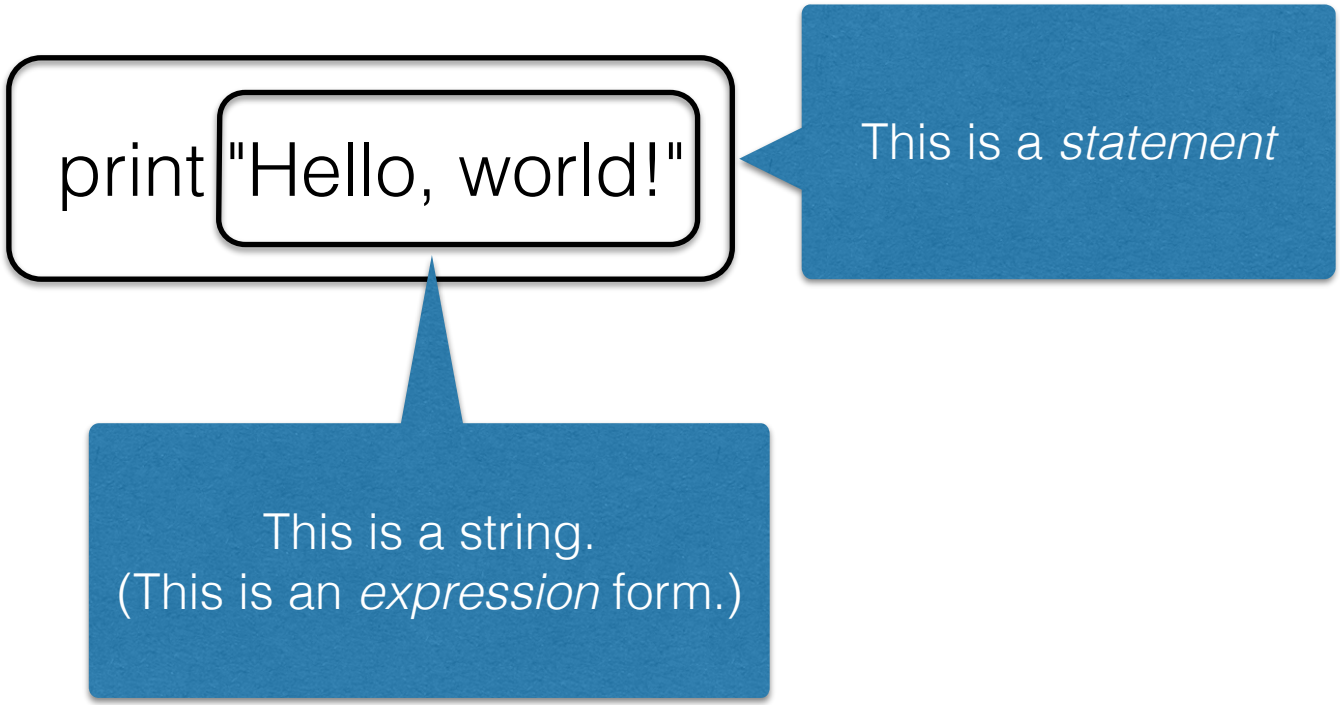
An executable software artifact that solves a certain problem

```
print "Hello, world!"
```

What's the problem solved by this Python script? :-)

© 2014, Ralf Lämmel, rlaemmel@acm.org

The hello world program:
it solves the problem of printing „Hello, world!“



© 2014, Ralf Lämmel, rlaemmel@acm.org

Another program

```
r = 1
for i in [1,2,3,4,5]:
    r *= i
print r
```

Indentation implies that this assignment is the body of the loop.

A for loop

What's the problem solved by this Python script?

© 2014, Ralf Lämmel, rlaemmel@acm.org

Another variation on ***5!***

```
r = 1
for i in range(1,6):
        r *= i
print r
```

Python language elements

- **Statements**
- **Expressions**
- **Literals (numbers, strings)**
- **Functions**
- **Comments**
- Classes with methods (functions)

© 2014, Ralf Lämmel, rlaemmel@acm.org

Python tutorial material

- The Python Tutorial: http://docs.python.org/2/tutorial/
- The LearnPython.org interactive Python tutorial: http://www.learnpython.org/
- A Beginner's Python Tutorial: http://www.sthurlow.com/python/

Perhaps, create a pythonanywhere account?

https://www.pythonanywhere.com/

pythonanywhere.com

- Go to pythonanywhere.com.
- **Sign up** for an account.
- **Create Beginner account**.
- Fill in all account data.
- Select „**I don't want any help! …**“.
- Select „**Files**“
- Create a „**src**“ folder.
- Create a file „**hello.py**“.
- Enter **print "hello"** into the file.
- Select „**Save and run**“.
- Enter **Ctrl D** to quit the console.

© 2014, Ralf Lämmel, rlaemmel@acm.org

The notion of *program*

An executable software artifact that solves a certain problem

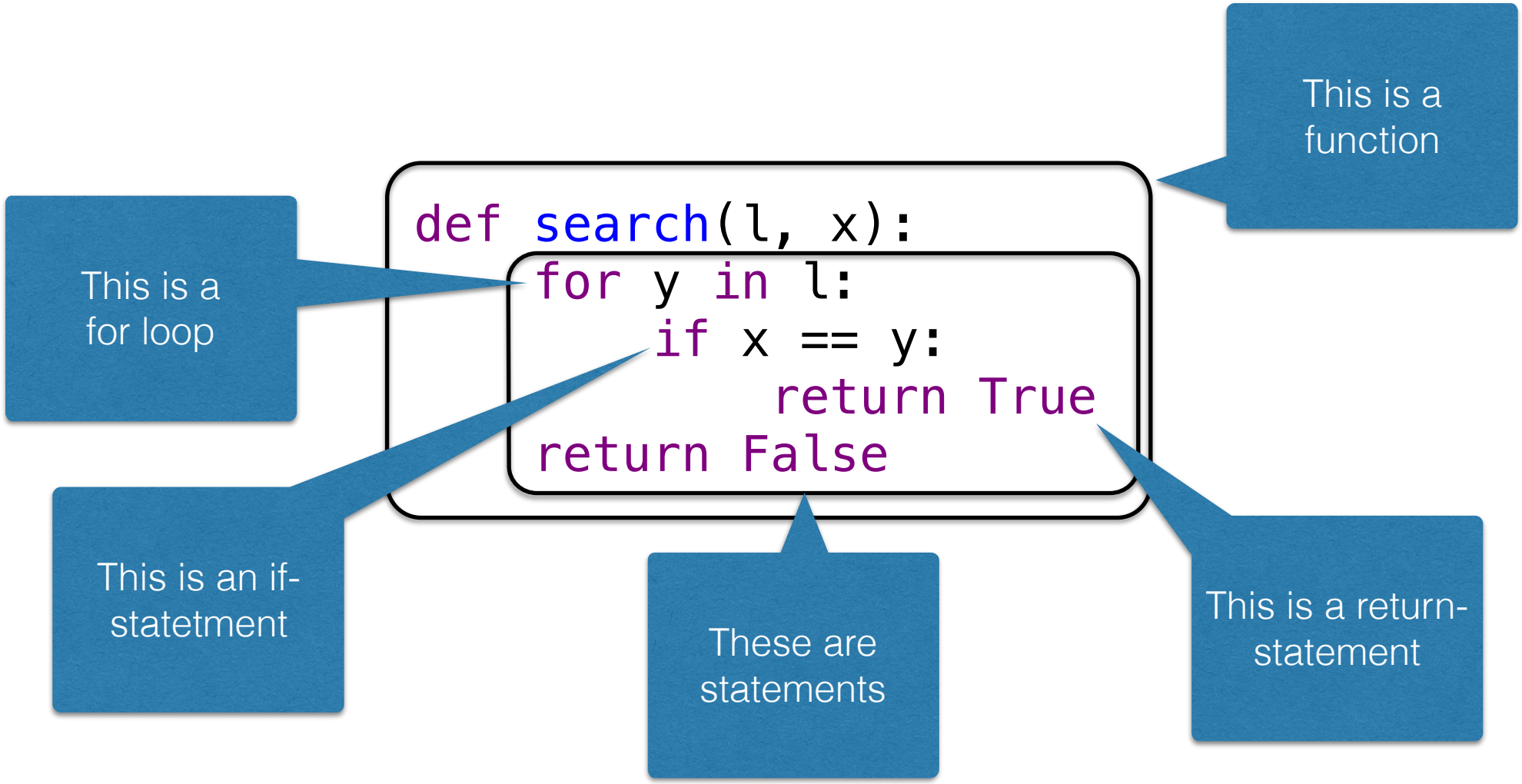
```
def search(l, x):
    for y in l:
        if x == y:
            return True
    return False
```

What's the problem solved by this Python script?

© 2014, Ralf Lämmel, rlaemmel@acm.org

http://101companies.org/wiki/Linear_search

Linear search



Python statements

- Assignments: x = 1
- If statements: if 1 == 1: print 42
- Return statements: return 42
- For loops: for i in [1,2]: print i
- …

© 2014, Ralf Lämmel, rlaemmel@acm.org

Testing linear search

```
input = [2,4,3,1,4]
print search(input, 1) # prints True
print search(input, 5) # prints False
```

Function application

Let's try at pythonanywhere. Please, try yourself now and later.

© 2014, Ralf Lämmel, rlaemmel@acm.org

What other ‚problems' do exist?

- Sort a list an ascending order.
- Computer the matrix product of two matrices.
- Delete a file in the file system.
- Transmit stock exchange request to a server.
- Provide a service accessible through a web server.
- …

© 2014, Ralf Lämmel, rlaemmel@acm.org

101companies.org/wiki/Algorithm

The notion of *algorithm*

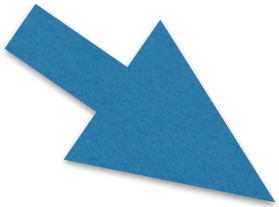
A step-by-step procedure for achieving a certain **IO behavior**

For example — Linear search:

- Given is a list *l* and a value *v* of the element type of *l*.
- Perform the following steps to search *v* in *l*:
  - Initialize an index variable *i* to 0 (to refer to the first element of *l*, if any).
  - Repeat the following steps until a result is returned.
    - Return *False*, if *i* equals the length of *l*.
    - Return *True*, if *l* stores *v* at the index *i*.
    - Increment *i*.

Algorithms are implemented as programs.

- Given is a list *l* and a value *v* of the element type of *l*.
- Perform the following steps to search *v* in *l*:
  - Initialize an index variable *i* to 0 (to refer to the first element of *l*, if any).
  - Repeat the following steps until a result is returned.
    - Return *False*, if *i* equals the length of *l*.
    - Return *True*, if *l* stores *v* at the index *i*.
    - Increment *i*.

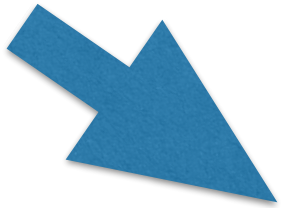


***Python* code**

```
def search(l, x):
    for y in l:
        if x == y:
            return True
    return False
```

Algorithms are implemented as programs.

- Given is a list *l* and a value *v* of the element type of *l*.
- Perform the following steps to search *v* in *l*:
  - Initialize an index variable *i* to 0 (to refer to the first element of *l*, if any).
  - Repeat the following steps until a result is returned.
    - Return *False*, if *i* equals the length of *l*.
    - Return *True*, if *l* stores *v* at the index *i*.
    - Increment *i*.

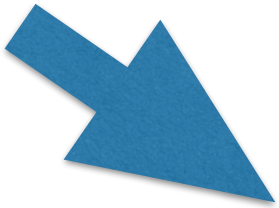


***Haskell* code**

```
search [] _ = False
search (x:xs) y = x==y || search xs y
```

Algorithms are implemented as programs.

- Given is a list *l* and a value *v* of the element type of *l*.
- Perform the following steps to search *v* in *l*:
  - Initialize an index variable *i* to 0 (to refer to the first element of *l*, if any).
  - Repeat the following steps until a result is returned.
    - Return *False*, if *i* equals the length of *l*.
    - Return *True*, if *l* stores *v* at the index *i*.
    - Increment *i*.



***Java* code**

```
public class LinearSearch {
    public static <T> boolean search(Iterable<T> l, T x) {
        for (T y : l)
            if (x==y)
                return true;
        return false;
    }
}
```

Why are there many languages?

- Language as a form of programming innovation.
- Language as a legacy in software development.
- Language as a cultural phenomenon.
- Static versus dynamic typing.
- Different programming paradigms.
- …

© 2014, Ralf Lämmel, rlaemmel@acm.org

101companies.org/wiki/Algorithm

The notion of *algorithm*

A step-by-step procedure for achieving a certain IO behavior

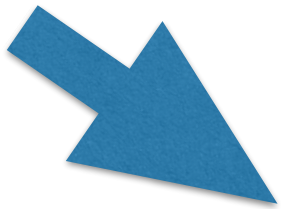
For example — Greatest common divisor:

- We assume (for simplicity) two positive operands *x* and *y*.
- Perform the following steps to return the greatest common divisor of *x* and *y*.
  - Repeat the following step until *x* equals *y*:
    - If *x* > *y*
      - then subtract *y* from *x* and
      - else subtract *x* from *y*.
  - Return *x*.

For example:
The GCD of 15 and 6 is 3.

Algorithms are implemented as programs.

- We assume (for simplicity) two positive operands *x* and *y*.
- Perform the following steps to return the greatest common divisor of *x* and *y*.
  - Repeat the following step until *x* equals *y*:
    - If *x* > *y*
      - then subtract *y* from *x* and
      - else subtract *x* from *y*.
  - Return *x*.

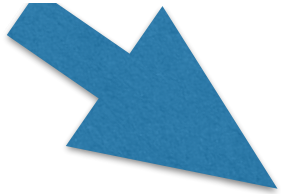


***Python* code**

```
def gcdIterative(x, y):
    while x != y:
        if x > y:
            x -= y
        else:
            y -= x
    return x
```

Algorithms are implemented as programs.

- We assume (for simplicity) two positive operands *x* and *y*.
- Perform the following steps to return the greatest common divisor of *x* and *y*.
  - Repeat the following step until *x* equals *y*:
    - If *x* > *y*
      - then subtract *y* from *x* and
      - else subtract *x* from *y*.
  - Return *x*.



Another *Python* code

Interesting use of recursion

```
def gcdRecursive(x, y):
    if x > y:
        return gcdRecursive(x-y, y)
    elif x > y:
        return gcdRecursive(x, y-x)
    else:
        return x
```

Does every program implement an algorithm?

- Programs may fail to solve the intended problem.
- Programs may be designed not to terminate.
- Programs may be designed to perform side effects.
- …

© 2014, Ralf Lämmel, rlaemmel@acm.org

Is the mapping *problem : program* a 1:1 mapping?

- Programs may solve problems in different ‚styles'.
- Programs may choose different pragmatics:
  - Use of external versus internal memory.
  - Use of encoding versus convenient representation.
  - …
- Programs may implement *refined* problems:
  - *Binary* search versus *linear* search.
  - …

Binary rather than linear search

- Suppose the input for search is *sorted*.
  - As in: [1,3,7,11]
  - As opposed to: [3,1,11,7]
- Search can eliminate half of input in each step.
  - Start in the middle
  - Go to the left, if value in the middle is larger than *x*.
  - Go to the right, if value in the middle is smaller than *x*.
  - Otherwise *x* has been found.

© 2014, Ralf Lämmel, rlaemmel@acm.org

http://101companies.org/wiki/Binary_search

Binary search

```
def search(l, x):
    return searchInRange(l, x, 0, len(l)-1)

def searchInRange(l, x, min, max):
    if min>max:
        return False
    else:
        middle = min+(max-min)/2
        if x > l[middle]:
            # Search in right half
            return searchInRange(l, x, middle+1, max)
        elif x < l[middle]:
            # Search in left half
            return searchInRange(l, x, min, middle-1)
        else:
            # Found in the middle
            return True
```

Forms of programming abstractions

- **Functions**
  - Examples: search, gcd, factorial
  - Abstractions for *expressions*
- **Classes** with methods
  - Examples: Account, Windows, Counter, Order
  - Abstractions for *capsules of data and behaviors*

© 2014, Ralf Lämmel, rlaemmel@acm.org

The notion of *data*

Quantities on which operations are performed in computing

```
>>> 1+1
2
```
Numbers and additions thereupon

```
>>> (True, 'True')
(True, 'True')
```
A pair of a Boolean and a String

```
>>> [1] + [2,3]
[1, 2, 3]
```
List and concatenation thereupon

```
>>> [(1,'2')]+[(3,'4')]
[(1, '2'), (3, '4')]
```
Lists of pairs

```
>>> salaries = dict()
>>> salaries['ralf'] = 42
>>> salaries['guido'] = 88
>>> salaries
{'ralf': 42, 'guido': 88}
```
Dictionaries

© 2014, Ralf Lämmel, rlaemmel@acm.org

The notion of *object*

Capsules of state and behavior in OO programming

```
>>> counter1 = Counter()
>>> counter1
0
>>> counter1.inc()
>>> counter1.inc()
>>> counter1
2
>>> counter1.reset()
>>> counter1
0
```

Create a counter

Show representation

Increment twice and show again

Reset counter and show again

© 2014, Ralf Lämmel, rlaemmel@acm.org

The notion of *object*

Capsules of state and behavior in OO programming

```
>>> cnum1 = Complex(1,2)
>>> cnum2 = Complex(3,4)
>>> cnum1.add(cnum2)
Complex(4, 6)
```

Create two complex numbers

Add and show result

© 2014, Ralf Lämmel, rlaemmel@acm.org

The notion of *object*

Capsules of state and behavior in OO programming

```
>>> l = [2,1,3]
>>> l.pop()
3
>>> l
[2, 1]
>>> l.append(4)
>>> l
[2, 1, 4]
>>> l.sort()
>>> l
[1, 2, 4]
```

Create a list

Remove the ‚last' element of the list

Add another element to the list

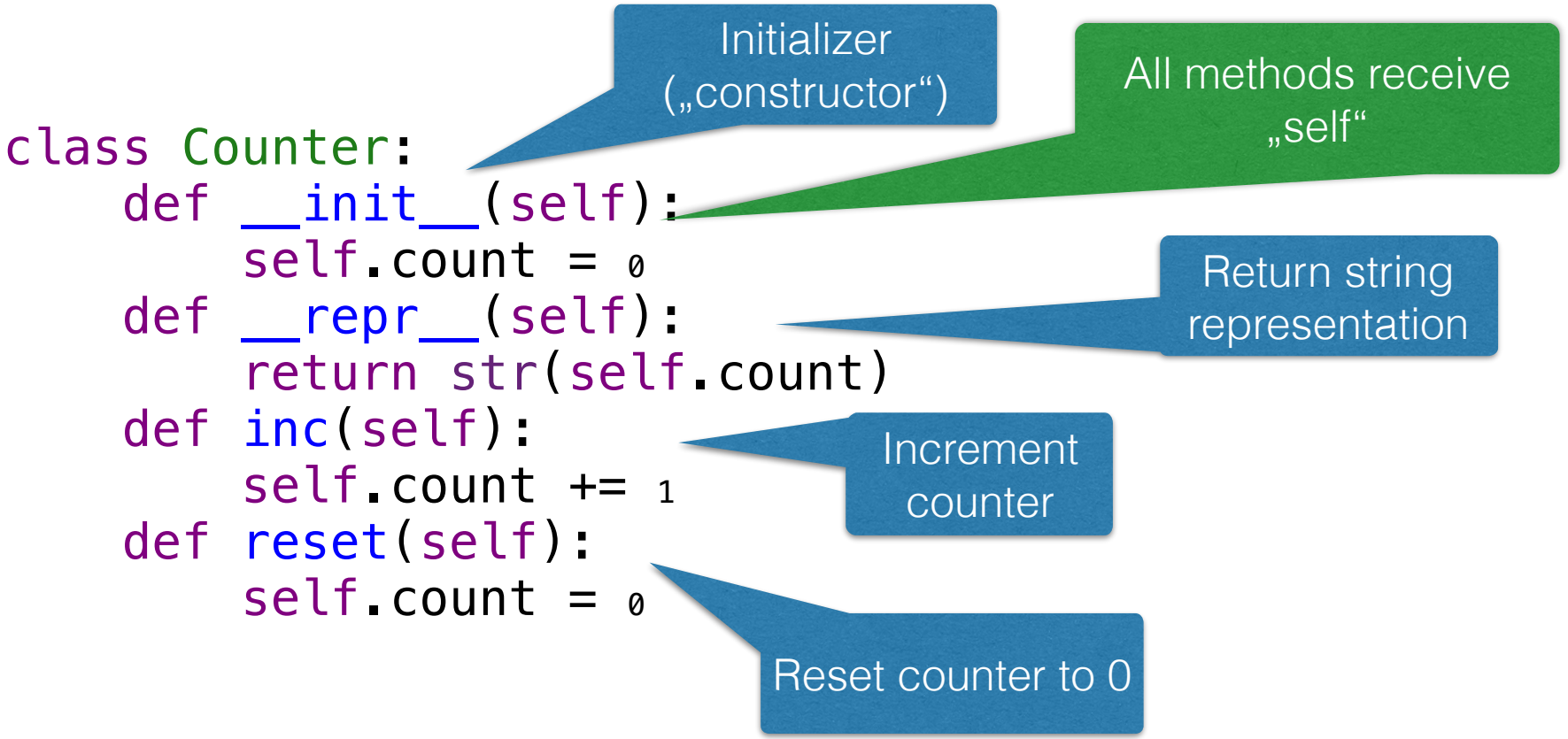
Sort the list

We don't care much how exactly those lists are internally represented. We basically care about the methods to **observe** and **manipulate** such **state**.

© 2014, Ralf Lämmel, rlaemmel@acm.org

The notion of *class*

An abstraction mechanism for objects




```
class Counter:
    def __init__(self):
        self.count = 0
    def __repr__(self):
        return str(self.count)
    def inc(self):
        self.count += 1
    def reset(self):
        self.count = 0
```

© 2014, Ralf Lämmel, rlaemmel@acm.org

The notion of *class*

An abstraction mechanism for objects

```
class Complex:
    def __init__(self, re, im):
        self.re = re
        self.im = im
    def __repr__(self):
        return "Complex(" \
               + str(self.re) \
               + ", " \
               + str(self.im) + ")"
    def add(self, cnum):
        return Complex(\
                  self.re+cnum.re, \
                  self.im+cnum.im)
```

Initializer („constructor“)

Return string representation

Addition of two complex numbers returns a new number as result.

© 2014, Ralf Lämmel, rlaemmel@acm.org

The notion of *API*

An interface for functionality that is reusable by programs

```
>>> import twitter
>>> api = twitter.Api(...)
>>> statuses = api.GetPublicTimeline()
>>> print [s.user.name for s in statuses]
[u'DeWitt', u'Kesuke Miyagi', u'ev', u'Buzz Andersen', u'Biz Stone']
>>> statuses = api.GetUserTimeline(user)
>>> print [s.text for s in statuses]
>>> users = api.GetFriends()
>>> print [u.name for u in users]
>>> status = api.PostUpdate('I love python-twitter!')
>>> print status.text
I love python-twitter!
```

Authentication details omitted

Retrieve and print public timeline

Retrieve and print timeline of user

Retrieve and print friends of user

Post a status update

Source: https://code.google.com/p/python-twitter/

© 2014, Ralf Lämmel, rlaemmel@acm.org

What other APIs can we think of?

- XML access
- Database access
- GUI widgets
- Stock exchange
- Your startup API
- …

GUI programming with Tkinter

http://101companies.org/wiki/Technology:Tkinter

```
ralfs-air:Tkinter laemmel$ python program.py
hi there, everyone!
hi there, everyone!
```

Wanted: the interactive „hello world“ script.

GUI programming with Tkinter

http://101companies.org/wiki/Technology:Tkinter

```
# Example from http://effbot.org/tkinterbook/tkinter-hello-again.htm

from Tkinter import *

class App:
    def __init__(self, master):
        frame = Frame(master)
        frame.pack()
        self.button = Button(
            frame, text="QUIT", fg="red", command=frame.quit
            )
        self.button.pack(side=LEFT)
        self.hi_there = Button(frame, text="Hello", command=self.say_hi)
        self.hi_there.pack(side=LEFT)

    def say_hi(self):
        print "hi there, everyone!"

root = Tk()
app = App(root)
root.mainloop()
root.destroy() # optional
```

How is programming more than writing code?

- *Design* your code.
- *Document* your code.
- *Test* your code.
- *Deploy* your code.
- *Evolve* your code.
- …

Some of these activities will be considered in this course eventually. **Design** was covered by the **modeling** discussion.