

Constructive Artificial Intelligence

Programming in Python

Daniel Polani

School of Computer Science
University of Hertfordshire

October 30, 2009

All rights reserved. Permission is granted to copy and distribute these slides in full or in part for purposes of research, education as well as private use, provided that author, affiliation and this notice is retained.

Use as part of home- and coursework is only allowed with express permission by the responsible tutor and, in this case, is to be appropriately referenced.

Example (Addition)

```
a = 3
b = 4
print a+b    # 7
```

Example (Exchange)

```
a = 3
b = 4
a, b = b, a    # exchange variables:
                # a becomes b and b becomes a,
                # simultaneously
```

- 1 A *condition* is specified by the keyword `if`, the boolean condition, a colon `:`, and a *block*
- 2 A block is a section of the code with additional indentation relative to the current code
- 3 If the condition applies, the block becomes active (i.e. the program runs through the block), if it does not, the block is skipped

Example

```
if x < 0:  
    print "negative"
```

Note

Alternative conditions are started with the keyword `elif`, and unconditional alternatives with the keyword `else`.

Example

```
if x < 0: print "neg"    # one-lined blocks can be kept
                        # on same line
elif x > 0: print "pos"
else: print "zero"     # if none of above cases
```

while Loops

- 1 A `while` loop is opened with the corresponding keyword, a condition, and a colon `:` and is followed by an (indented) block
- 2 the block is repeated until the condition becomes `False`

Example

The following example prints 0,1,...,10

```
i = 0
while i <= 10:
    print i
    i += 1          # increment i by 1
```

Function Calls

Function Definition:

```
def <function name>(<arguments>): <block>
```

Example (Sorting by Direct Comparison)

```
def sorter(x,y):  
    return min(x,y), max(x,y)
```

Note

`min` and `max` are predefined functions, taking arbitrary arguments, and returning the corresponding value

Method Calls: call a method for a given data structure via the usual `<data structure>.<method>()` notation.

Example (Sorting)

```
mylist = [5,2,1,6,2]
```

```
mylist.sort()    # sort IN PLACE!
```

Sorting Arguments: `cmp`, the comparator function can be overridden.

Example (Customized Sorting)

```
# cmp(x,y) as used by the list.sort() method returns
# -1 if x < y,
# 0 if x = y
# 1 if x > y

def even(x):
    return x % 2 == 0 # '%' is modulo/remainder operator

def odd(x):
    return not even(x) # better to define by existing functions

def odd_even_cmp(x,y):
    """Sort first all odd numbers first, then even."""
    if odd(x) and even(y): return -1
    if even(x) and odd(y): return 1
    return cmp(x,y)

mylist = [5,2,1,6,5,2,3,6]
mylist.sort(cmp=odd_even_cmp)
print mylist          # prints [1, 3, 5, 5, 2, 2, 6, 6]
```

Exercise 1

Assumptions

You have four numbers (integers or floats) x, y, z, u .

Task

Write a function `sort4(.)` that takes 4 numerical arguments and returns a sorted tuple of these arguments. Do not use the list's `sort()` method.

List Comprehensions

Neat Trick: List comprehensions are a neat way of defining lists almost like sets.

Example

```
[i*i for i in range(5)] # creates [0, 1, 4, 9, 16]  
[i for i in range(5) if even(i)] # creates [0, 2, 4]
```