

# Constructive Artificial Intelligence

## Introduction

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.

# Python: the Tail of the Snake

## Creating Variables:

- variables created at first definition
- syntax: `<variable name> = <value>`
- type of variable content can vary

### Example

```
num = 1                # setting num to 1
motto = "Creating AI"  # setting motto to string
motto = 42             # *the* answer
```

# Python: the Tail of the Snake

## Creating Variables:

- variables created at first definition
- syntax: `<variable name> = <value>`
- type of variable content can vary

### Example

```
num = 1                # setting num to 1
motto = "Creating AI" # setting motto to string
motto = 42             # *the* answer
```

### Note

- 1 no variable declaration
- 2 num is of type `int(eger)`
- 3 motto is of type `str(ing)` at first definition/assignment
- 4 motto becomes of type `int` at second assignment
- 5 comment starts with `#` until end of line

## Possible Operations:

- int and str (and some other) objects can be added using +

### Example

```
num = 3 + 3           # stores 6 in num
s = "bli" + "bla"    # stores "blibla" in s

num += 4              # add 4 to whatever
                     # is already in num,
                     # i.e. num is 10 now
```

# Operating on Strings

- 1 strings are a sequence of characters
- 2 characters counted from 0 to  $n$  where  $n$  is the length of the string
- 3 access the characters of a string  $s$  via square brackets  $s[\dots]$

# Operating on Strings

- 1 strings are a sequence of characters
- 2 characters counted from 0 to  $n$  where  $n$  is the length of the string
- 3 access the characters of a string  $s$  via square brackets  $s[\dots]$
- 4 start counting characters from 0 (*not 1*)!

## Example

```
s = "abcdef"
print s[0]           # prints 'a'
print s[2]           # prints 'c'
print s[-1]          # prints 'f'
print s[-2]          # prints 'e'
```

# Operating on Strings II

- 1 can extract not just characters, but also subsequences of characters (*substrings*)
- 2 for this, use the ':' syntax
- 3 specify starting character on left
- 4 specify character beyond ending character on the right

## Example

```
s = "abcdef"
print s[0:2]           # prints 'ab'
print s[1:4]          # prints 'bcd'

print s[3:-1]         # prints 'de'
                      # starting at 4th character,
                      # ending before
                      # (excluding) the last
```

# Operating on Strings III

Note: For default entries, do not specify indices

## Example

```
s = "abcdef"

print s[:2]           # prints 'ab' from the beginning
                     # to before index 2

print s[3:]          # prints 'def'
                     # from index 3 to end

print s[-2:]         # prints 'ef'
                     # from second last to end

r = s[:]             # r contains now a copy of s
print len(s)         # prints 6, the length of s
```

- a list can contain a sequence of various objects

## Example

```
["foo", "bar", "bli", "bla"] # all strings
[1, 5, 6, 7]                 # all integers
["foo", "bar", 1, 5]        # mixture of int and str

["foo", 1, 2.3, [5, 6, 7]]  # a str, an int, a float
                             # and a list of three int
```

# Lists II

- 1 list elements are addressed very similarly to string elements
- 2 they are counted starting from 0
- 3 elements from the end are addressed with negative indices
- 4 sublists can be addressed by the ':' notation

## Example

```
mylist = [1,2,3,4,5,6]

print mylist[1]           # prints '2'!
print mylist[-1]         # prints last, i.e. 6
print mylist[2:4]         # prints sublist [3,4]

mylist[2:4] = ["bla"]     # replace the sublist with
                          # 3 and 4 by the list with
                          # just "bla" - what
                          # if you forget the []
                          # around "bla"?

print mylist              # prints [1,2,"bla",5,6]
```

# Lists III: Some Extras

Let `mylist=[1,2,3,4,5,6]` as above.

## Default Parameters

An empty first parameter to `:` defaults to beginning of list, empty last parameter to end of list (as in `str`)

## Example

```
mylist[:3] # gives [1,2,3]
mylist[4:] # gives [5,6]
mylist[:]  # copies mylist
```

## Concatenation

Adding lists concatenates them

## Example

```
[1,2,3]+[4,5,6] # [1,2,3,4,5,6]
```

## Lists IV: Some Extras

### List Length

`len(.)` returns the length of the list (the dot stands for the list inserted into the expression)

### Example

```
len(mylist) # 6
```

### Operations

One can operate on elements of a list

### Example

```
mylist[2] += 5 # adds 5 to 3rd element, giving  
# [1,2,8,4,5,6]
```