GETTING STARTED WITH PYTHON

20-11-2012
Philo van Kemenade
CAST Sandbox
Goldsmiths College

HELLO

Philo van Kemenade

BSc Artificial Intelligence, University of Amsterdam

MSc Cognitive Computing, Goldsmiths College

I use Python for:

tracking video propagation in over 100gb of tweets

building statistical models for natural language processing

analysing experimental data from a database

talking to robots :)

You can find these slides at:

tinyurl.com/usingpython

THIS SESSION

- Intro
- Python Syntax
  - Variables and data types
  - Collections of Data
  - Functions & Methods
  - Flow of Control
- Exercise 1
- Writing scripts
- Importing libraries
- Exercise 2
- Useful References

INTRO: about programming...

What is programming?

INTRO: about programming...

Misconceptions:

- only ‘coders’ can program
- something magical is happening
- programming is difficult

Take home

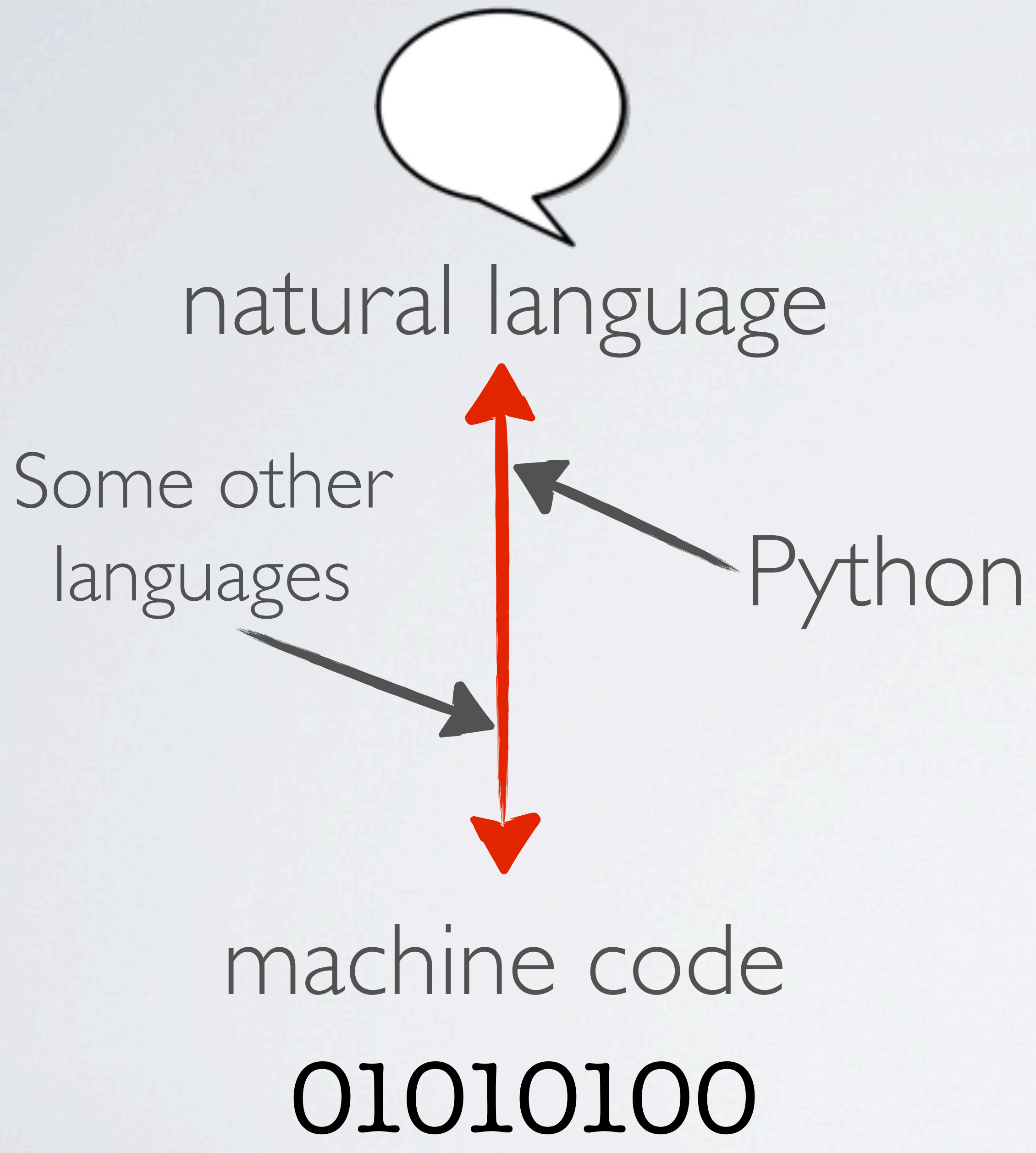
- programming language ~= natural language
- programming needs active exercise
- no secret magic
- you need to start somewhere

INTRO: programming a script

Telling your computer what to do

Statements are elementary instructions that make up a program

INTRO: programming with Python



one of Python's most attractive features

INTRO: programming with Python

Python:

- Easy to use
- Powerful & fast
- Connects to other languages and protocols
- Platform independent
- Strong community
- Free & open source

INTRO: Python

Two modes of Operation

Interactive Mode
Terminal-based
Direct feedback
Let’s you try out code

Scripting Mode
Code in file
Save & load programs
More control

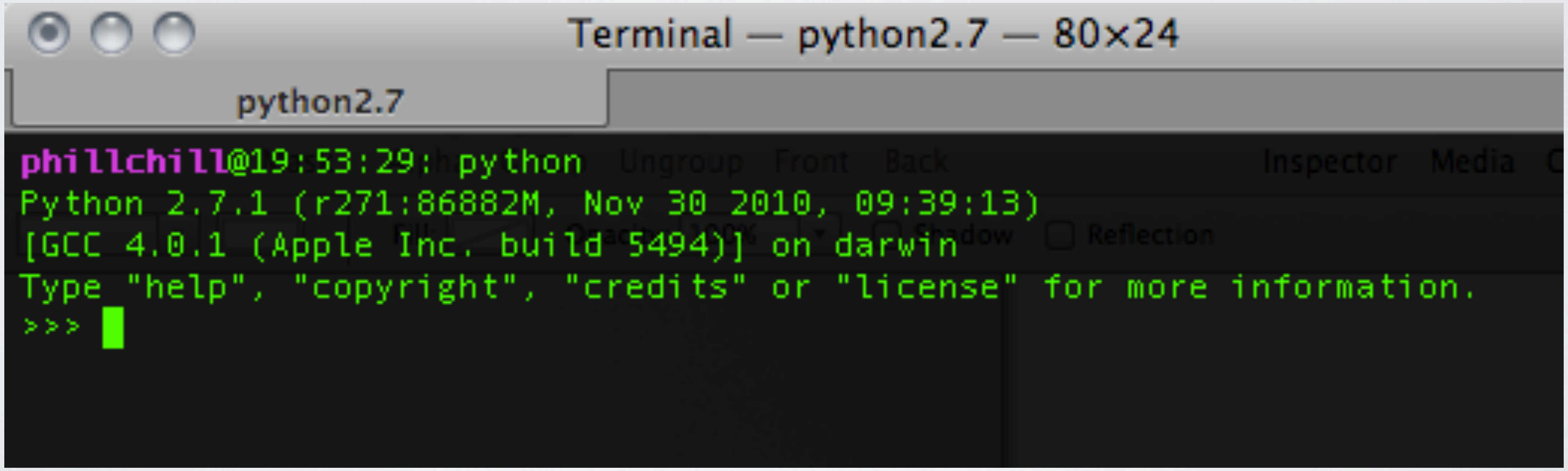
INTRO: Setting up

Download python 2.7 from http://python.org/download/

Install python

Open terminal / cmd window

Type "python" to start Python



```
phillchill@19:53:29: python
Python 2.7.1 (r271:86882M, Nov 30 2010, 09:39:13)
[GCC 4.0.1 (Apple Inc. build 5494)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```

VARIABLES & DATA TYPES

Variables store data under any specified name.

Data can be of different *types*:

`int` - an *integer*, or whole number [1,5,9999, ...]

`float` - a *floating point* number (using a decimal point) [3.14, 1.68, 0.1, ...]

`bool` - *boolean*; binary true or false values [True, False]

`string` - a *sequence of characters*, comprising text ['a', 'goldsmiths', 'asparagus']

OPERATORS

You can process the data in your variables by ***operators***:

For example:

=	assignment: assign a value to a variable
==	comparison: are two variables equal?
!=	comparison: are two variables unequal?
<, >, <=, >=	less-than, greater-than, less or equal, greater or equal
+, -, *, /	mathematical operators
&, |	logical operators and, or

EXAMPLE

```
>>> a = 5
>>> a + 2
7
>>> a
5
>>> a = a * 2
>>> a
10
>>> b = a * a
>>> b
100
>>> c = a + b
>>> c
110
>>> first = "gold"
>>> last = "smiths"
>>> first + last
'goldsmiths'
>>> 3 * s + t
'goldgoldgoldsmiths'
```

EXERCISE

Can you make a sentence by using *strings* stored in *variables* s & t?

What happens if we compare s and t with the '<'or '>' *operators*?

Why?

COLLECTIONS OF DATA

Data can also be stored in a *collection*:

- List
- Dictionary
- Tuple
- Set

COLLECTIONS OF DATA

Data can be stored in a ***list***:

```
>>> l = [1,3,9,4,884328881]
>>> n = ['sex', 'drugs', 'rock', 'roll']
>>> m = l + n
>>> m
[1, 3, 9, 4, 884328881, 'sex', 'drugs', 'rock', 'roll']
```

A *list* is a sequence of items (between [...]) that all have their own index:

```
>>> m[0]
1
>>> m[7]
'rock'
```

COLLECTIONS OF DATA

Data can also be stored in a ***dictionary***:

```
>>> nowdict = {'location':'goldsmiths college',
'activity':'CAST workshop', 'temperature':20}
>>> nowdict['location']
'goldsmiths college'
```

Dictionaries are collections of (*key:value*) pairs (between {...}).

Values are indexed by a unique string or integer (the *key*)

Dictionary items are unordered

EXERCISE

Create a dictionary for a specific class, include information like 'classname', 'roomnumber', 'lecturer', etc.

Add a list of students to your dictionary. What do you use as *key*, what do you use as *value*?

FUNCTIONS

Functions perform multiple tasks, collected under a specific name

Take input (one or more *argument(s)*), return output

Input and output can be of all different types

Recognizable by pair of parentheses

For example:

```
>>> name = "Philo van Kemenade"
>>> length = len(name)
>>> print(length)
18
>>> type(length)
<type 'int'>
```

METHODS

A ***method*** is a kind of function, belonging to particular *object*.
Think of it as taking the object (before '.') as an argument.
Methods can return different data types

For example:

```
>>> name.isupper()
  False
>>> name.upper()
  'PHILO VAN KEMENADE'
>>> 'www.example.com'.strip('cmowz.')
'example'
>>> nowdict.keys()
['location', 'temperature', 'activity']
>>> nowdict.values()
['goldsmiths college', 20, 'CAST workshop']
```

FLOW OF CONTROL: Loops

You can use *loops* to repeat a statement.
A ***for-loop*** is useful when you know how many times you want to repeat an action (e.g. for every item in a list)

For example: (note the *indentation*)

```
>>> boringlist = [1,2,3,4]
>>> for number in boringlist:
...     print(number)
...
1
2
3
4
```

Tip: use range([number]) to create a sequence from 0 until [number]

FLOW OF CONTROL: Loops

A ***while-loop*** is useful when you don't know when you want to stop looping yet.
A while-loop statement checks a condition and loops until the condition is no longer satisfied.

For example: a three year-old simulator

```
>>> while ans != 'because!':
...     ans = raw_input("why?\n")
...
why?
because you're not old enough
why?
because it's too late
why?
because!
>>>
```

FLOW OF CONTROL: Conditional statements

A ***conditional statements*** enable you to deal with multiple options. You can perform conditional checks with:

`if, (elif), else`

For example:

```
>>> boringlist = [1,2,3,4]
>>> for number in boringlist:
...     if number > 2:
...             print(number)
...     elif number < 2:
...             print("you're too small")
...     else:
...             print("2 is a nice number")
...
you're too small
2 is a nice number
3
4
```

SYNTAX SO FAR

Questions?

EXERCISE 1

Can you make a sentence by using *strings* stored in *variables* s & t?

What happens if we compare s and t with the '<' or '>' *operators*?

Why?

(Cheating is encouraged)

EXERCISE 2

Create a *dictionary* for a specific class, including information like 'classname', 'roomnumber', 'lecturer', etc.

Add an entry to your dictionary that contains a list of students

Use a *function* to calculate the number of students

Bonus: use a *loop* to print all the students

(Cheating is encouraged)

QUESTIONS

?

WRITING SCRIPTS

You can also structure your code conveniently in a file

Such a file is a ***program*** or ***script*** and can look as simple as this:

```
print("Hello World")
```

Save your script as “whatever_clear_name**.py**”

Navigate in terminal to the location of your file

Run “`python whatever_clear_name.py`”

WRITING SCRIPTS

Or this:

```
# this is a comment

'''
Comments can also
span multiple lines
'''

# print() is a very useful function
# mind the quotes
print("Hello World")

# you can also use "print [what you want to print]" without parentheses
print "Hello Sun"
```

WRITING SCRIPTS

You can define your own functions:

```
# define a function
def print_stuff():
    # create a new dictionary
    nowdict = {'location':'London', 'temperature':20}
    # print some info
    print "dict: ", nowdict
    print "length: ", len(nowdict)
    print "keys: ", nowdict.keys()
    print "values: ", nowdict.values()

# call a function
print_stuff()
```

WRITING SCRIPTS

You can pass in command line arguments

```
# import sys module for access to command line arguments
import sys

# define main function
def main(argv):
    # check if the list argv has 1 argument
    if len(argv) == 1:
        # use first argument from argv in print statement
        print "Hello " + argv[0]
    # otherwise exit with instructions
    else:
        exit("Please specify exactly one argument")

# start executing
# checking if __name__ is equal to "__main__" is a trick to also be able to
#   call the script in interactive mode (not important for now)
if __name__ == "__main__":
    # call main function, pass command line arguments
    main(sys.argv[1:])
```

IMPORTING LIBRARIES

A ***library*** is a package of code that extends the native functionality of Python. Use libraries to:

- plot graphs
- open URLs
- use the Twitter API
- read and write .csv files
- ...

Importing a library is simple:

```
>>> import pprint
>>> uglydict = {'one':range(10), 'two':range(15)}
>>> pprint.pprint(uglydict)
{'one': [0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
 'two': [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]}
```

EXERCISE 3

Write a script that:

constructs a dictionary of animals

- where each animal is represented by a dictionary containing its characteristics

prints some basic info about:

- the dictionary as a whole
- the animals in the dictionary

USEFUL RESOURCES

www.google.com; "python" + your problem / question

www.python.org/doc/; official python documentation, useful to find which functions are available

http://docs.python.org/tutorial/; official tutorial if you want to explore more detail

www.stackoverflow.com; huge gamified forum with discussions on all sorts of programming questions, answers are ranked by community

http://tinyurl.com/usingpython; these slides :)