Programming with Python

Vedran Šego <vsego@vsego.org>

29.1.2015.

MANCHESTER 1824
The University of Manchester

M\crNA
Manchester Numerical Analysis

Outline

Outline

Who? What? Where?

- Name: Vedran Šego
- Office: 1.218
- E-mail: vsego@vsego.org
  Please use:
  - only plain text (no HTML nor non-standard characters),
  - clear, informative subjects.
- Office hours: Friday, 12:00–13:00
- The course website:
  http://vsego.org/math20622 or http://vsego.org/python

Outline

Where can I use any of this?

While we shall cover none of these topic, you should become able to "dig out" on your own how to tackle the problems in:

1. Matrix Analysis (MATH36001) – SciPy and NumPy modules;
2. Combinatorics and Graph Theory (MATH39001) – modules `math`, `itertools`, `combinatorics`, `pyncomb`, `qombinatorics`, `networkx`, built-in lists and dictionaries (see here);
3. Mathematical Biology (MATH35032) – `SciPy` module for ODEs, `PyMOOSE` module, built-in lists and dictionaries;
4. Problem Solving by Computer (MATH36032) – Python and MATLAB are quite similar (and somewhat a competition);
5. Mathematical Programming (MATH39012) – modules `PyLPSolve`, `CVXOpt`, `PyGLPK`, `PyMathProg`.

And these are just some of the 3rd year undergraduate courses.

In this course...

...we aim to learn how to:

1. analyze a problem,
2. construct an algorithm that solves it,
3. write a program that implements the said algorithm.

In this course...

...we aim to learn how to:

1. analyze a problem,
2. construct an algorithm that solves it,
3. write a program that implements the said algorithm.

Be careful!

- Where do I expect "trouble"?
  Point 2: the so called *algorithmic way of thinking*.

In this course...

...we aim to learn how to:

1. analyze a problem,
2. construct an algorithm that solves it,
3. write a program that implements the said algorithm.

Be careful!

- Where do I expect "trouble"?
  Point 2: the so called *algorithmic way of thinking*.
- Why?
  This is fundamentally new to those with no experience with programming.

In this course...

...we aim to learn how to:

1. analyze a problem,
2. construct an algorithm that solves it,
3. write a program that implements the said algorithm.

Be careful!

- Where do I expect "trouble"?
  Point 2: the so called *algorithmic way of thinking*.
- Why?
  This is fundamentally new to those with no experience with programming.

Our language of choice: Python 3 (with some effort, fairly simple!)

However,...

...we do **not** aim to:

- make you into World class programmers (this would take at least a whole programme and does not fit in a single course),
- address all the features and the libraries of Python (this would take even longer),
- learn "dark" tricks that almost all languages have.

However,...

...we do **not aim** to:

- make you into World class programmers (this would take at least a whole programme and does not fit in a single course),
- address all the features and the libraries of Python (this would take even longer),
- learn "dark" tricks that almost all languages have.

Our aim is "only" to lay a strong foundation for your future programming requirements, be they in Python or some other language or a similar tool.

However,...

...we do **not aim** to:

- make you into World class programmers (this would take at least a whole programme and does not fit in a single course),
- address all the features and the libraries of Python (this would take even longer),
- learn "dark" tricks that almost all languages have.

Our aim is "only" to lay a strong foundation for your future programming requirements, be they in Python or some other language or a similar tool.

Don't worry, this is not as easy as it sounds. ☺

Father and son

Python currently has two branches:

- Python 2: old(-ish), well established, widely used;
- Python 3 (our choice): newer, better designed, also widely used, gaining acceptance, some modules are not converted (yet), but almost all widely used either are converted or will soon be.

The two are very similar, but there are some significant differences!
$\Rightarrow$ Some of these will be addressed as necessary.

Father and son

Python currently has two branches:

- Python 2: old(-ish), well established, widely used;
- Python 3 (our choice): newer, better designed, also widely used, gaining acceptance, some modules are not converted (yet), but almost all widely used either are converted or will soon be.

The two are very similar, but there are some significant differences!
$\Rightarrow$ Some of these will be addressed as necessary.

Our focus is on algorithms, not on the programming language!
$\Rightarrow$ Python specifics will be avoided as much as possible.
(but they will be shown for those interested in Python itself)

Lectures and lab classes

- Week 1:
  - This lecture: about the course.
  - Lab class (tomorrow): the first lecture.
- From the second week on:
  - Lectures: covering the new material.
  - Lab class: students solve problems on computers.

Marking

- Six one-hour tests during the lab classes (30%):
  - The best five will be taken into account.
  - The first one is in week 3.
  - Two problems on each; 3 points per problem.
  - Small(-ish) programs done on computers.
  - Covering the material up to the previous week.
- Coursework at the end of semester (70%):
  - A project to program at home.

Outline

How to tackle this course?

Programming is a skill. It cannot be learnt; it has to be crafted.

— prof. Saša Singer, University of Zagreb

How to tackle this course?

Programming is a skill. It cannot be learnt; it has to be crafted.
— prof. Saša Singer, University of Zagreb

This means:

- Traditional "learning by reading" is of little use.
- Trial-and-error on a computer is the only way to learn!
- Programming cannot be learnt the night (or even a day or two) before the exam!

How to tackle this course?

Programming is a skill. It cannot be learnt; it has to be crafted.
— prof. Saša Singer, University of Zagreb

This means:

- Traditional "learning by reading" is of little use.
- Trial-and-error on a computer is the only way to learn!
- Programming cannot be learnt the night (or even a day or two) before the exam!

Why?

- Remember the "fundamentally different algorithmic way of thinking" from a few slides before?
- Too many details to consider – **experience is crucial!** (and you're here to get some of it)

How to actually do this? (1)

- Copy/paste (from the ready made examples and solutions) is **not** your friend! If you must use such a solution, retype it, so you notice the important details!

How to actually do this? (1)

- Copy/paste (from the ready made examples and solutions) is **not** your friend! If you must use such a solution, retype it, so you notice the important details!
  For example, these four are different types of data, mostly with very different properties:

```
f = ( "x", 17, "y", 19, "z", 23 )
b = [ "x", 17, "y", 19, "z", 23 ]
r = { "x", 17, "y", 19, "z", 23 }
j = { "x": 17, "y": 19, "z": 23 }
```

  Will you notice and remember the type of brackets or which of the comma/colon was used if you just copy/paste a program with only one of these?

How to actually do this? (1)

- Copy/paste (from the ready made examples and solutions) is **not** your friend! If you must use such a solution, retype it, so you notice the important details!
  For example, these four are different types of data, mostly with very different properties:

```
f = ( "x", 17, "y", 19, "z", 23 )
b = [ "x", 17, "y", 19, "z", 23 ]
r = { "x", 17, "y", 19, "z", 23 }
j = { "x": 17, "y": 19, "z": 23 }
```

  Will you notice and remember the type of brackets or which of the comma/colon was used if you just copy/paste a program with only one of these?
  How about the difference between these two?

```
f = h
g = h(x)
```

How to actually do this? (2)

- Using a solution from a lecture or some other source is fine a first few times when you encounter a new subject, but even then do **NOT** just retype.
  Run the program and test it (give it some input and check that the output is correct).

How to actually do this? (2)

- Using a solution from a lecture or some other source is fine a first few times when you encounter a new subject, but even then do **NOT** just retype.
  Run the program and test it (give it some input and check that the output is correct).
  Take a break from it and, some hours later, try to write your own solution to the same problem, without looking at or trying to recall the "official" solution.

How to actually do this? (2)

- Using a solution from a lecture or some other source is fine a first few times when you encounter a new subject, but even then do **NOT** just retype.
  Run the program and test it (give it some input and check that the output is correct).
  Take a break from it and, some hours later, try to write your own solution to the same problem, without looking at or trying to recall the "official" solution.
- **Do NOT learn chunks of code by heart!**
  These cannot be just "reused"; they must be **understood**!
  Otherwise, you get a wrong impression of "understanding", which backfires in exams and practical situations.

How to actually do this? (3)

Write programs on your own!

How to actually do this? (3)

Write programs on your own!

- Read the errors and warnings that Python writes out. These are crucial to understand what went wrong!

How to actually do this? (3)

Write programs on your own!

- Read the errors and warnings that Python writes out.
  These are crucial to understand what went wrong!
- As before, test your program! Run it several times, give it some input, and check the results!
  If they are wrong, read your code and try to figure which part(s) produce a result different from what you expected.

How to actually do this? (3)

Write programs on your own!

- Read the errors and warnings that Python writes out.
  These are crucial to understand what went wrong!
- As before, test your program! Run it several times, give it some input, and check the results!
  If they are wrong, read your code and try to figure which part(s) produce a result different from what you expected.
- Test the boundary cases!
  Does your expansion to prime factors work if the input itself is a prime number? What if it's a negative number?
  Does your algorithm work properly with the first/last member of the list, or only with the ones in the middle? How about an empty list?
  ...

How to actually do this? (4)

What if it doesn't work out?

How to actually do this? (4)

What if it doesn't work out?

0. Try until you run out of ideas.
1. If you cannot make the program work as it should, read the solution and analyze it. Try to figure out how it works, how it was constructed, and why your own didn't work.
2. Then return to it later and try to solve the problem without using the solution (reading Python documentation is always fine).
3. If that fails, go back to step 1.
4. If you have to do this for (almost) all problems, seek help with the staff. Do not wait for the exams to address the issue! This process takes time and effort!

How to actually do this? (4)

What if it doesn't work out?

0. Try until you run out of ideas.
1. If you cannot make the program work as it should, read the solution and analyze it. Try to figure out how it works, how it was constructed, and why your own didn't work.
2. Then return to it later and try to solve the problem without using the solution (reading Python documentation is always fine).
3. If that fails, go back to step 1.
4. If you have to do this for (almost) all problems, seek help with the staff. Do not wait for the exams to address the issue! This process takes time and effort!

Remember: Your solution need not be the same as the "official" one, but it has to produce the correct results.
(Almost) all problems have many (more or less different) solutions.

How to actually do this? (4)

What if it doesn't work out?

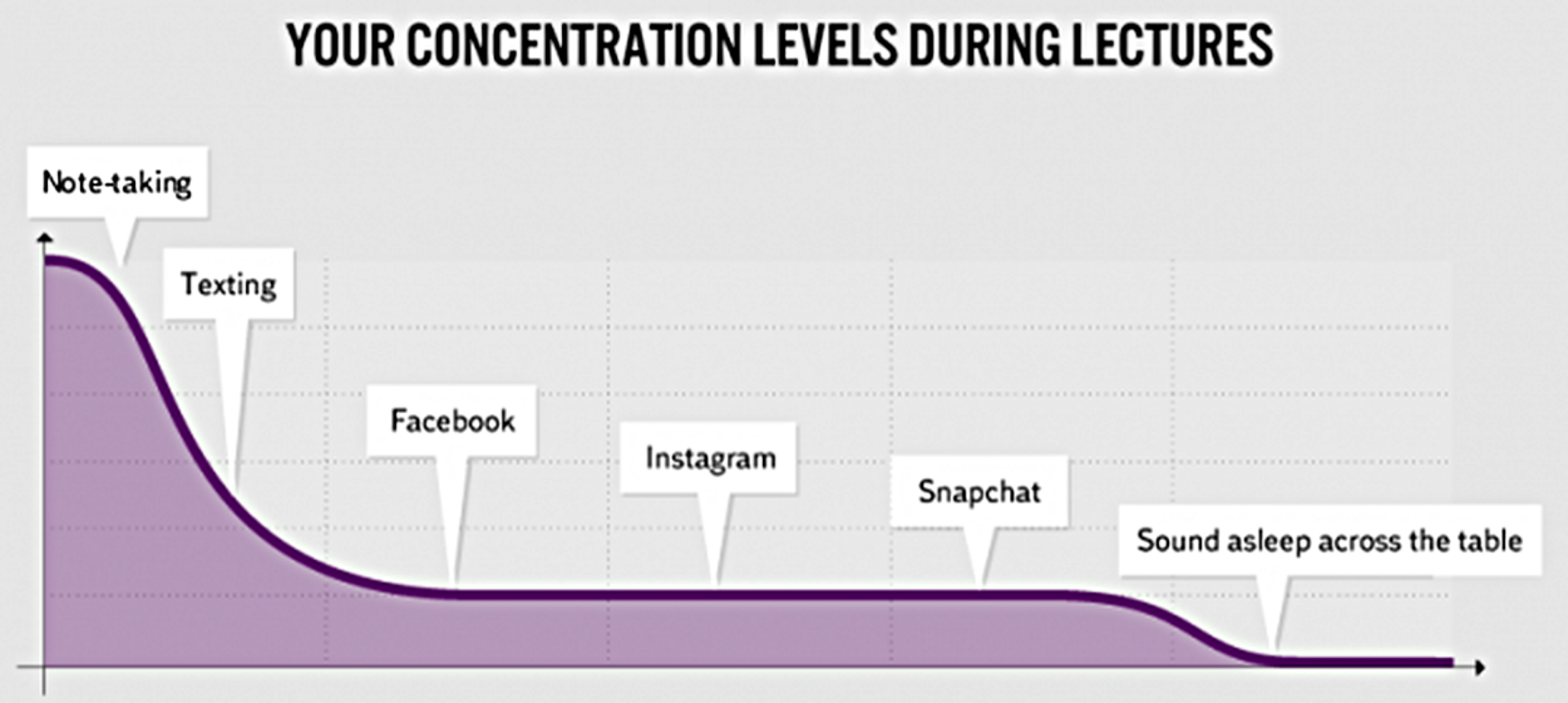
0. Try until you run out of ideas.
1. If you cannot make the program work as it should, read the solution and analyze it. Try to figure out how it works, how it was constructed, and why your own didn't work.
2. Then return to it later and try to solve the problem without using the solution (reading Python documentation is always fine).
3. If that fails, go back to step 1.
4. If you have to do this for (almost) all problems, seek help with the staff. Do not wait for the exams to address the issue! This process takes time and effort!

Remember: Your solution need not be the same as the "official" one, but it has to produce the correct results.
(Almost) all problems have many (more or less different) solutions.

By the way, the above is an example of an algorithm. ☺

How to actually do this? (9 3/4)

How you approach these classes might also have a wee bit of an impact on the whole "crafting your programming skills" process... ☺



[Image source]

Outline

Asking for help (1)

If you encounter a problem, address it immediately – most of the lectures rely on the understanding of the previous ones!

You can ask:

- teaching assistants,
- me,
- your colleagues (be careful, though),
- on-line.
  I suggest Stack Overflow. Before asking a question there, read their instructions! The community is very helpful, but they require effort and will not just solve your problems for you.

Asking for help (2)

At a more advanced level:

- Code Review Stack Exchange – get expert opinion and advice on your properly working code;
- Programmers Stack Exchange – for conceptual questions about software development.

As before, read their instructions before asking a question!

Asking for help (2)

At a more advanced level:

- Code Review Stack Exchange – get expert opinion and advice on your properly working code;
- Programmers Stack Exchange – for conceptual questions about software development.

As before, read their instructions before asking a question!

Hint: There is also Mathematics Stack Exchange...

References (1)

Some useful references:

- For most of the questions, it is enough to Google `python3 whatever you want to know`
- The official Python 3 documentation (there is also a Python 2 documentation, if you ever need it),
- Built-in help, invoked from Python itself:

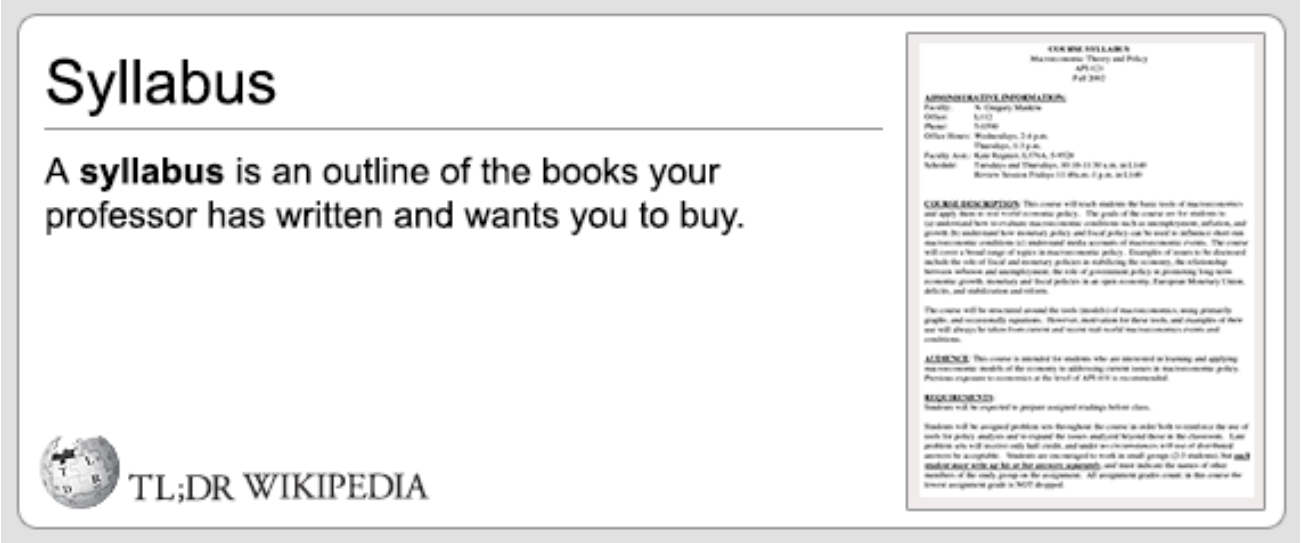
```
help(print)
```

  or

```
import numpy
help(numpy)
```

- Aforementioned Stack Exchange sites.

References (2)



For those preferring dead wood books,

- Mark Lutz, "Learning Python". (no, I am not Mark Lutz ☺)

For those already familiar with programming, just not in Python:

- Mark Pilgrim, "Dive into Python 3" (freely downloadable HTML and PDF); you can also buy a dead wood version, or install it for free on Android devices.
  There is a Python 2 version as well.

Homework

- Install Python 3 (the instructions are on the course web site) on your home computer.
- Try running this program to verify that your Python installation works properly:

```
import this
```

  Yes, there is only one line and there are no interpunctions.
  It should print a poem "The Zen of Python", by Tim Peters.

Homework

- Install Python 3 (the instructions are on the course web site) on your home computer.
- Try running this program to verify that your Python installation works properly:

```
import this
```

  Yes, there is only one line and there are no interpunctions.
  It should print a poem "The Zen of Python", by Tim Peters.
  And yes, there is some weird humor in Python. After all, it was named after (somewhat famous) Monty Python.

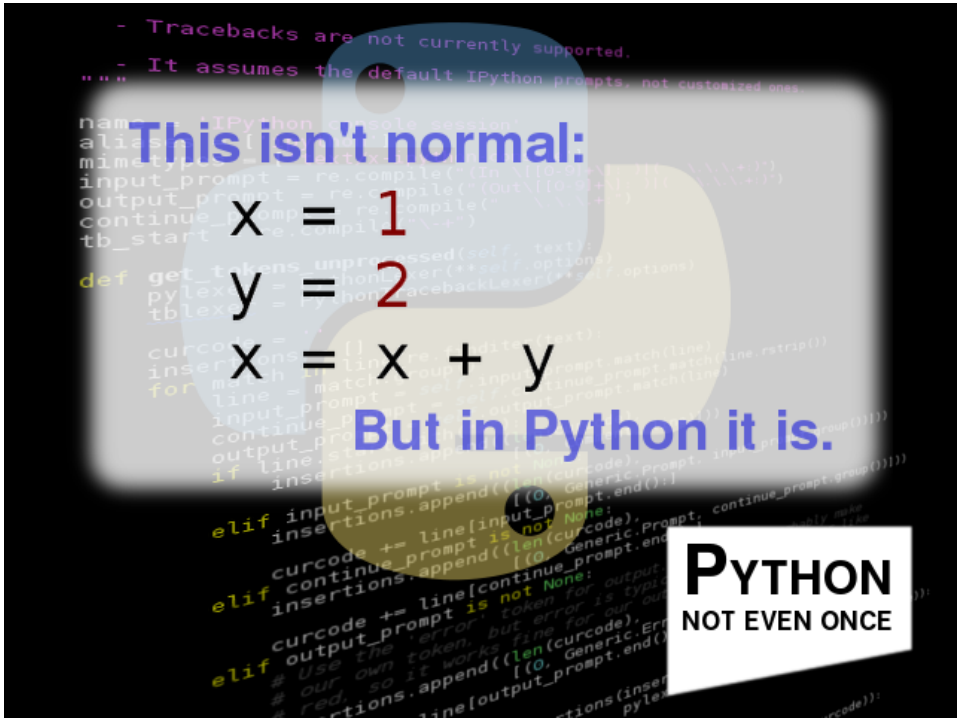
Homework

- Install Python 3 (the instructions are on the course web site) on your home computer.
- Try running this program to verify that your Python installation works properly:

```
import this
```

Yes, there is only one line and there are no interpunctions.
It should print a poem "The Zen of Python", by Tim Peters.
And yes, there is some weird humor in Python. After all, it was named after (somewhat famous) Monty Python.
But don't worry, none of it affects "real" programming...

That's all for today (unless, of course, there are questions)



Thank you for your attention!