Introduction
First Steps in Python

Intro2CS – week 1a
The Course Staff

The Lecturers
• Aviv Zohar
• Noam Nisan

Tsars:
• Yaacov Weiss
• David Zisselman

Teaching Assistants:
• Ohad Dan
• Guy Eyal
• Ayal Green
• Shelly Mahlab
• Asaf Valadarsky
• Doron Zarchy

+ Exercise graders
+ Lab support
Where to find all needed information

www.cs.huji.ac.il/~intro2cs

(Leads to the moodle webpage of the course)

exercises, lecture slides, office hours, forums, course email, etc.
Open an account

If you do not yet have an account:

Go to  http://w.cs.huji.ac.il

Enter your ID number and open an account.

• Pick username and password carefully!
• If something does not work: contact the System group: system@cs.huji.ac.il
Weekly schedule

• Every week we have
  – 3 hours of lectures (2 on Sunday and 1 on Thursday)
  – 2 hours of tirgul

• Expect an exercise to work on every week. Programming takes time especially if you have little or no prior experience.

• Two exercises are already online!
  – Ex0: Due this Wednesday (Open an account first!)
  – Ex1: Due next Wednesday.
What to expect

• A programming exercise approx. every week (50% of final grade)

• A written exam at the end of the course (50% of final grade)

• Must have passing grade (at least 60) in both to pass course
Make sure you read the course policy on the website!

• This course has many students, things will run smoothly only with your help.

Know where to direct your questions/requests:
• Personal issues → email intro2cs@cs.huji.ac.il
• Questions about exercises → forums.
• For questions regarding the material: We also have office hours (Teachers and TAs) & lab support (at the labs).
Every student must solve the *targil* on their own.

على كل סטודנט לפתור את התרגיל בעצמו.
You are not allowed to be told what to write in a program, or to read someone else’s code.
What IS Allowed?

• Discussion is allowed, even encouraged, but **without looking at code**!
  – You are required to list students you discussed and ex with (in README file of each exercise).

• How do you know if you are doing something wrong? **If you are looking at someone else’s code or someone is looking at yours, this is definitely not allowed.**

• If someone copies from you, **both of you are responsible!**
What is this course about?

This course aims to give you

- basic programming skills
  Python 3

- foundations of computer science theory
  Algorithms and their formal analysis.

a taste of things that you can do with programming
  Image processing, Connecting on the Internet, Writing Games, GUIs, etc.
# Course schedule

(approximate)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction, functions, variables, types, expressions</td>
</tr>
<tr>
<td>2</td>
<td>Conditionals, Strings, Parameters, loops</td>
</tr>
<tr>
<td>3</td>
<td>More loops and Lists</td>
</tr>
<tr>
<td>4</td>
<td>Aliasing, scope, mutability &amp; Collections</td>
</tr>
<tr>
<td>5</td>
<td>Numeric algorithms, Runtime analysis</td>
</tr>
<tr>
<td>6</td>
<td>Simple Search &amp; Sorting</td>
</tr>
<tr>
<td>7</td>
<td>Recursion</td>
</tr>
<tr>
<td>8</td>
<td>Object Oriented Programming</td>
</tr>
<tr>
<td>9</td>
<td>Linked data structures (Lists, Trees, etc)</td>
</tr>
<tr>
<td>10</td>
<td>iterators/generators. 2nd order programming</td>
</tr>
<tr>
<td>11</td>
<td>more 2nd order, event-driven programming &amp; GUI</td>
</tr>
<tr>
<td>12</td>
<td>Optimization local vs global, brute force search</td>
</tr>
<tr>
<td>13</td>
<td>More optimization, Formal models &amp; languages</td>
</tr>
<tr>
<td>14</td>
<td>More formal models, Halting and Goedel</td>
</tr>
</tbody>
</table>
Introduction
Early “computers” were mechanical they could add, subtract, multiply, etc. They were not programmable.
Colossus

- Helped crack German codes during WW2
- ~1500 vacuum tubes

- Previous computing machines were mechanical
ENIAC

• Electronic Numerical Integrator And Computer

• Built to calculate firing tables for the US army

• Constructed 1946
Semi Conductors

- Switch faster than vacuum tubes
- Transistors can be “printed” on silicon

Intel 4004 – 4bit CPU (1971)

Intel core i7 (quad core)
Moore’s law (1965)
Prediction: Number of transistors will double every 2 years. (Actually doubles every 18 months)
Gordon E. Moore (co-founder of Intel)
The Internet

• The second revolution in computing
• Connect to computers all around the world.
• Information is highly accessible, easily and cheaply transmitted.

• ubiquitous computing + internet -> “Big Data”
Programming

• Access to all this computing power and massive amounts of information is literally at your fingertips.

• The way to access it all is to learn to write code.
Computer Programs & Algorithms

• Computer programs are basically lists of instructions for the computer to follow.

• They use basic building blocks: commands and other structures that are defined by the language.

• Programs Must be precise!

• Algorithms: being clever can give incredible advantages.
How many people in this room?

Initialization:
• Everyone stands up
• Person sitting in lowest row on the right gets the value 1. Starts “executing”.

Person who is currently “executing”:
• Call out your number
• Pass your number +1 to standing person nearby.
• Sit down
Algorithms

How long does this take?

Can we speed up the process?
Counting (version 2)

Initialization:
• Everyone stands up
• Everyone starts with the value 1

While you are still standing:
• Pair up with another person who is standing
• One of you adds your numbers together, keeps the result
• The other one sits down
Problem size (n)

Running time

n

n/2

log(n)

Problem size (n)
Questions we will answer: how exactly to think about running time.

– What happens when same program runs on different computers, is it faster?

Are there other costs besides time?

What is “the best” algorithm for the job?

How do we know?
First Steps in Python
Compiled vs. Interpreted Programs

• Computers only run programs written in low-level machine code.
  – This code is different for each type of computer.
  – It is very hard for humans to read or to program.

• To make things easy: program languages are more human readable. Programs are then converted to this code

```python
print(1234+4321)
```
Compiled vs Interpreted Languages

Program Code (text file) $\rightarrow$ Compiler $\rightarrow$ Compiled Executable Program (machine code)

Then, run the machine code.

Pros: Compilation can take place far in advance, can optimize code
Cons: Compiled code fits specific machine. Every change to code requires compilation again

Program Code (text file) $\rightarrow$ Interpreter (Runs Program)

No separate compilation stage. Program runs.

Pros: agile development
Cons: no time for optimization and for lots of checks on the code.
Programming in Python

• We will be learning Python 3
  – (There is a huge difference between python 2.x and python 3.x)

• Python is an interpreted language.
• We therefore use the python interpreter to run programs.

• There are versions for Windows, Mac, Linux, etc.
Pycharm + WinPython

• If you are working on a windows PC we recommend that you install

• WinPython (a bundle that has an interpreter + a lot of packages + and IDE called Spyder)

• Pycharm (yet another IDE) can also be installed. This is the IDE that is set up on the lab computers
Where to find help when programming

• Try things out. You won’t break anything.

• Google is your friend.

• Tutorials and docs on python website

• Books.
  – Example: “Think Python, How to Think Like a Computer Scientist”. Online (for python 3) here: http://faculty.stedwards.edu/mikek/python/thinkpython.pdf
The First Python Program

• The first python command we will use is print()
• In a text file called hello_world.py

print("Hello World!")

• Your TAs will show you this again in Tirgul
• The command `print()` will print text to the screen.

• Running the program:
• Things to notice:
  – print() should be lower case. This is important in general. The computer considers lower /upper case as different symbols.
  – No white space at the beginning of the line
  – Printed text can be lower case or upper case, as you wish.
• **print()** is a *function*. It can take several *arguments*, separated by commas

\[
\text{print("hi", "there")}
\]

which prints:

\[
\text{hi there}
\]

(notice the space between the words)
• Programs are composed of *statements*
• We can write several *statements*, each in a new line, which will be executed in sequence:

```python
print("one", "two", "three")
print("four")
print("five")
```

• Which prints:

```plaintext
one two three
four
five
```
• Code can be hard to read. To make programs clear, we can add comments

```python
print("one", "two", "three")
# a comment that is ignored.
print("four")
print("five") # another comment
```

• Good comments help to explain code.
• You are required to write code that is commented – a good habit.
• Due to constrained space on slides, code examples will not always be well documented.
Python’s Interactive Interpreter

• In addition to programs written in files Python also has an “interactive” interpreter
• Type in a command, and it will be evaluated, and printed.
• Useful for checking behavior of small bits of code, but not for long programs.
• It can also evaluate math operations:

![Python Shell]

```
>>> 2+4
6
>>> 10-3*5
-5
>>> 3.7/2.53
1.4624505928853757
>>> 
```
Expressions

• We can use more complex expressions to get python to do calculations for us:

```python
print(2+4)
print(10-3*5)
print((10-3)*5)
print(3.7/2.53)
```

• The expressions are evaluated, then printed

<table>
<thead>
<tr>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>-5</td>
</tr>
<tr>
<td>35</td>
</tr>
<tr>
<td>1.4624505928853757</td>
</tr>
</tbody>
</table>

Things to notice:

- We are not printing text. (No quotes “”)
- Operator precedence.
Other math operations

• Python has some functions that can perform other operations:

```python
>>> min(10, 2, 3)
2
>>> max(10, 2, 3)
10
>>> abs(-13)
13
>>> |
```

• They too can be used in expressions
We can create our own functions:
Small set of instructions that performs a task that we will use elsewhere in the program

```
def func_name():
    statements
    return
```

- Function name: lower case letters separated by underscore
- Indentation: 4 spaces
- Parentheses ( ) are a must
- After definition takes effect, we can use the function anywhere in code
functions

• We can create our own functions

```python
def myFunc():
    print("**********")
    print("* HELLO *")
    print("**********")
    return

myFunc()
```

**********
* HELLO *
**********

Process finished with exit code 0
A theme in computing

• Take basic building blocks and Build something more complicated (but useful).
• Package it as an advanced building block
• Start working with the advanced blocks...
(We can forget the internal complexity)
Turtle

```python
import turtle

turtle.forward(100)
turtle.left(120)
turtle.forward(50)
turtle.right(90)
turtle.backward(50)
turtle.done()
```
```python
import turtle

def petal():
    turtle.forward(100)
    turtle.left(120)
    turtle.forward(50)
    turtle.right(90)
    turtle.backward(50)

def flower():
    petal(); petal(); petal()
    petal(); petal(); petal()
    petal(); petal(); petal()
    petal(); petal(); petal()
```