Extended Introduction to Computer Science
CS1001.py

Lecture 1: Introduction and Administratrivia
First Acquaintance with Python

Instructors: Benny Chor, Amir Rubinstein
Teaching Assistants: Amir Gilad, Michal Kleinbort

Founding Teaching Assistant (and Python Guru): Rani Hod

School of Computer Science
Tel-Aviv University
Spring Semester, 2017
http://tau-cs1001-py.wikidot.com

Travel Advisory

- You are about to take the "extended introduction to CS" course.
- It is intended primarily for first year Computer Science students.
- The programming language taught and used in the course is Python (version 3.6).
- The course will require a substantial amount of work by yourselves to digest and pass successfully.

- Welcome aboard!

Administrative Details

- Grade determined by exam (85%) and homework (15%).
- In order to pass the course, you must pass the exam (grade at least 60), and also get a non failing grade (at least 50) in all home assignments.
- Homework grade is a weighted average of all n home assignments grades: Equal weight to the best $n-1$ grades, plus half the weight for the one worst grade.
- Moed A exam will be on July 28th, 2017 (Moed B on September 7th).
- Exam is closed book except for 2 double sided, normal size (A4) pages.

Administrative Details (2)

- A correctly filled form regarding course regulations ("thou shalt not copy", etc.) will contribute 2 points to the homework portion of the grade.
- 5-6 home assignments, each with both “dry” and “wet” component.
- It is mandatory to submit all homework assignments and get a grade of at least 50 in each of them.
- The homework should be submitted individually.
- Regulations regarding homework submissions will be published in the course site. We closely follow them, and you should carefully read them.
- Textual parts should be typed and follow length restriction (on number of words and lines).

Contact Information

- Office hours (till further notices): By e-appointments.
- E-mails: benny AT cs.tau.ac.il , amirr AT tau.ac.il , amirgilad2233 AT gmail.com , balasmic AT tau.ac.il .
- Course site, including a forum: http://tau-cs1001-py.wikidot.com

Administrative Details (3)

- In each "normal" week (not counting holidays, strikes, ceremonies, wars, etc.) there will be two lectures (2 hours each).
- In each "normal" week, there will be one recitation (2 hours as well).
- Lectures are given on Sundays (2 meetings) and Wednesdays (2 meetings) by the two lecturers.
- Recitations are given on Mondays and Thursdays by the two teaching assistants (over 4 meetings).
- You can choose lecture slot and recitation (priority given to students registered at group).
- The material covered during the two Sunday lecturers, the two Wednesday lectures, and each of the four weekly recitations is meant to be the same (note: Sundays $\neq$ Wednesdays).
- The homework and final exam are identical for all groups and subgroups.

Collaboration on Assignments, etc.

- Preparing homework assignments independently is a key ingredient for understanding the material (and, consequently, passing the exam :-). So it is highly recommended you make a serious effort to solve the problems on your own.
- You may (read "are encouraged to") discuss, consult, collaborate with other students on the problem sets, but your solutions must be written up independently, by you alone.
- You are welcome to consult online and offline sources for your solutions, but you are (a) expected to give clear cites of your references, and (b) use a write up of your own.
- Recall that Google (or any alternative search engine) is a double edge sword.

Collaboration on Assignments, etc.

- Cases of plagiarism that will be detected will be dealt with severely. (For example, reducing your grades for the whole course, not just the relevant assignment, and/or reporting the incident to the appropriate university authority.)
- If we suspect Alice had copied from Bob, both will be regarded as cheaters.

Course Structure

- We will "sample" 10–12 different topics of interest from a fairly wide range in Computer Science.
- This should hopefully expose you to some of the beautiful topics and ideas in the field.
- Naturally we will not get into any of them in great depth.
- We leave this to the required and elective courses you will take later in your studies.
- Each topic will be accompanied by programming examples (given by us) and tasks (to be done by you).
- Lectures and recitations are coordinated, and form integral parts of the course.
- Lectures in the same day by both lecturers are in principle the same, as are the 4 recitations (by the two teaching assistants) on the same week.

Programming Language Aspects

- This is not a programming course. Yet, you will learn and use a specific programming language – Python.
- And basic ideas in programming like iteration, control structure, recursion, procedures, objects, basic data structures, etc. etc.
- Python is a relatively new programming language, gaining popularity in many applications.
- As of July 2014, Python is the most popular language for teaching introductory computer science courses at top-ranked U.S. universities. Specifically, eight of the top 10 CS departments (80%), and 27 of the top 39 (69%), teach Python in their introductory courses. (source: CACM blog by Philip Guo).
- At TAU, we have moved to Python (from Scheme) 6 years ago. The Hebrew University followed suit, 3 years later. Even the hard core bastion known as Technion was considering such a move.

- Those of you who heard about Scheme and were hoping to learn it here, are a few years late :-(and counting)

Prior Knowledge

- Students in the course come from a wide range of backgrounds.
- This is especially true regarding prior exposure to programming.
- We designed the course so students who have a lot of programming experience will still learn a lot from it.
- And yet, students with no computation/programming background should be able to excel in it.
- This is a challenge for both the teaching staff and the students.
- Students with no programming background will probably be required to work harder, esp. during the first half of the course.
- We have allocated three weekly hours by an experienced student for help with programming and other course related issues.
- Time and venue: to be announced.
- You are strongly encouraged to take advantage of this ongoing oppurtunity.

One More Thing this Course is Not About

Course Structure

- Python programming basics (2–3 meetings)
- Bits, bytes, and representation of numbers in the computer.
- Huge integers, with applications to public key cryptography.
- Sorting and Searching.
- Basic data structures, incl. hashing and hash functions.
- Numerical computations (Newton–Raphson root finding).
- String matching, with applications in computational biology.
- Text compression.
- Representing and manipulating images.
- Simple error correction codes.
- Problems that cannot be solved by any computer.
- Hard computational problems.

Course Site: Lecture Notes and Forum

Lecture notes in pdf format (Adobe Acrobat) and Python code used in the class will be published prior to the lecture.

Likewise, Python code and logs of running the code in the recitation will be posted on the course site.

The course site will also host a forum, where students are welcome and encouraged to raise issues related to material taught in classes and in recitations.

Attending Lectures, YouTube, etc.

- Fall 2011-12 lectures by Benny Chor were video taped, and are available on youtube.
- This year's lectures will not be video taped.
- This year's lectures will be fairly similar to the 2011-12 year's lectures. But there are significant differences in some topics, while others are quite similar.
- You are welcome to watch these videos. We think they may help you digest the material at home. But they do not provide a complete replacement for this year.

- Attending lectures or recitations is not obligatory.
- Yet, we believe it gives you the unique opportunity to ask questions, and even get answers (feature still missing from the current implementation of youtube videos, moocs, etc.)

And Now For Something Completely Different

This type of transparency indicates we are moving to a new topic. It can be your chance of re-joining the lecture flow, in case you got lost.

Bibliography

There are many intro to CS textbooks and even online courses available. But our course does not use any of them exclusively. You are encouraged to primarily use the course slides and pointers to relevant bibliography.

There are also many Python references, but many of them are in fact manuals for the language.
Two notable exceptions, and one recommended manual, are:

1. Think Python, by Allen B. Downey, which is available online.
2. A book by John Zelle, "Python programming: an introduction to computer science", second edition. Fraklin, Beedle & Associates.
   The second edition refers to Python 3.x, which is the version used in the course.
   A few copies of this book have already found their way to the library (519.836 ZEL).
3. Python 3.6 documentation, http://docs.python.org/py3k/, is the official language manual, and a very useful resource.

Preface

Many of you have had extensive programming experience in various languages (C, C++, C#, Java, Perl, Pascal, Algol, Fortran, Basic, Cobol, Lisp, etc.) and in different contexts.

Of those, many have probably written some code in Python.

Yet many others in the audience have had little or no programming experience, except maybe the "intro to intro programming", given at the end of last summer (2016).

The senior lecturer highly sympathizes with the latter group (having started his own university studies, sometime in the early dawn of the programming era, with very little programming experience, in Fortran, gained as a side effect of a high school mathematics project).

So we will start from the very bare basics of computer languages in general, and Python programming in particular.

Programming Languages Basics

A computer program is a sequence of instructions (texts) that can be "understood" by a computer and executed by it.
In some sense, a computer program resembles a recipe for preparing food.
Pots, ovens, and even the final consumer of food, are typically quite tolerant. Putting a bit more sugar or a little less nutmeg will hardly be felt.
By way of contrast, an extra parenthesis, or a missing colon or quotation marks, will most likely cause a program to crash.

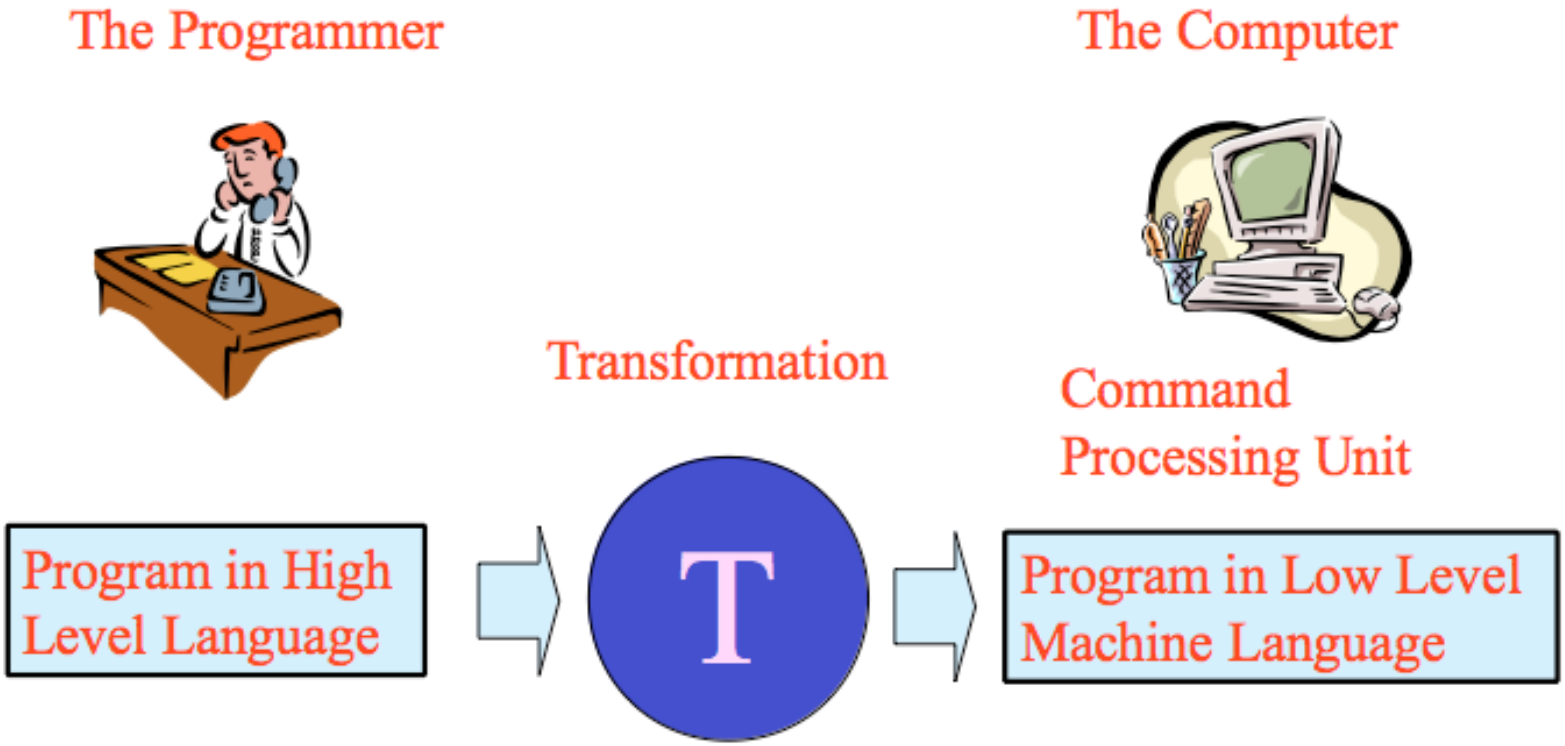
Writing Programs

- Getting a program to work as planned is an interesting process.
- It can often be not just interesting, but frustrating as well.
- Planning what your program should do, and how it is going to do it, is crucial.
- It is very tempting to skip such planning and go straight to writing lines of code.
- When things go wrong, it is even more tempting to change a line of the code and hope this will solve the problem.
- We strongly advise you not to skip the planning stage (both before and during the process).

From High Level to Machine Level Languages

Most programs these days are written in high level programming languages. These are formal languages with strict syntax, yet are fairly comprehensible to experienced programmers.

By way of contrast, the computer hardware "understands" a lower level machine code. The high level language is transformed to the machine code by yet another computer program.



(figure taken from the Scheme course site)

The Two Flavors of this Transformation

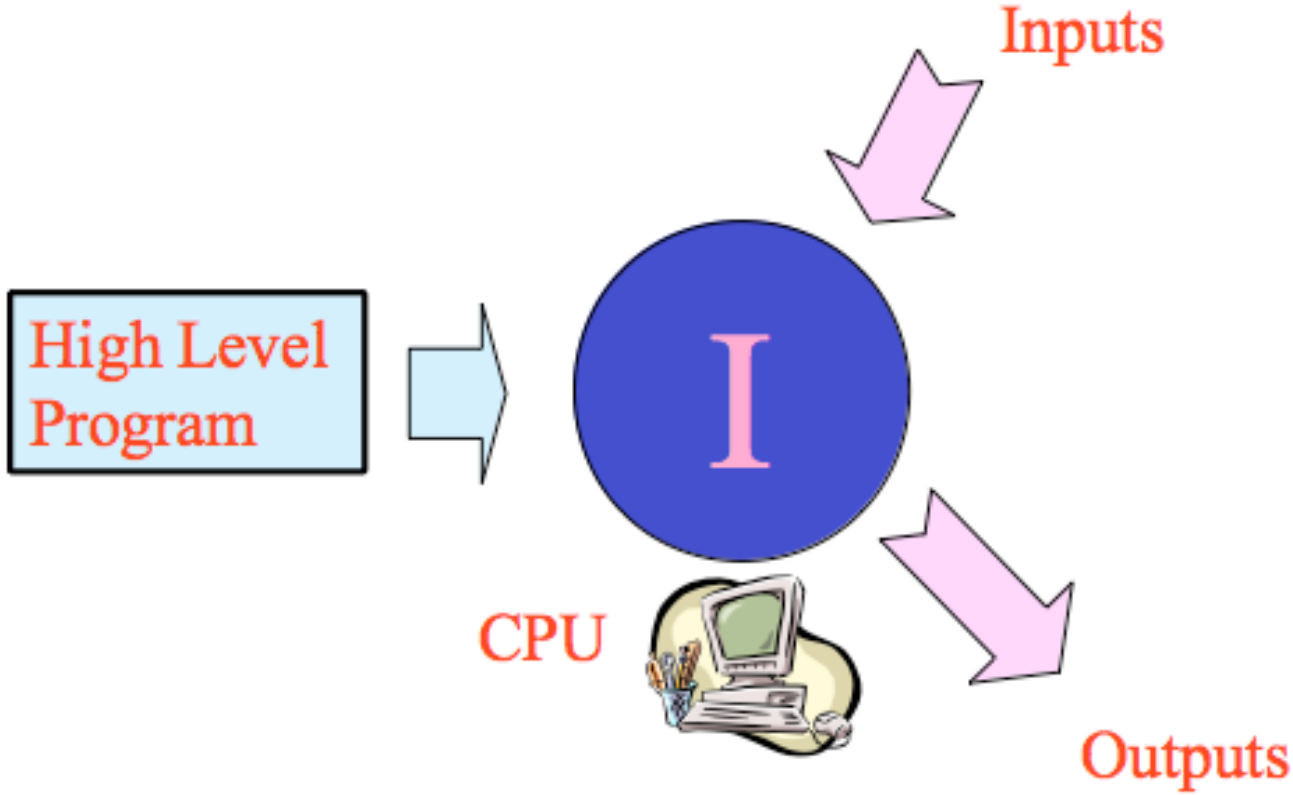
The transformation from high level to machine level languages comes in two flavors: By interpreters, and by compilers.

Python is an interpreted programming language.
And so are JavaScript, Lisp (and its variant, Scheme), MATLAB, Perl, PhP, and many many other programming languages.

In contrast, Java is a compiled programming language.
And so are C, C++, Fortran, Haskell, Pascal, Ruby, and many many other programming languages. (More precisely, Java is compiled to "bytecode", which is then interpreted.)

The Interpreter

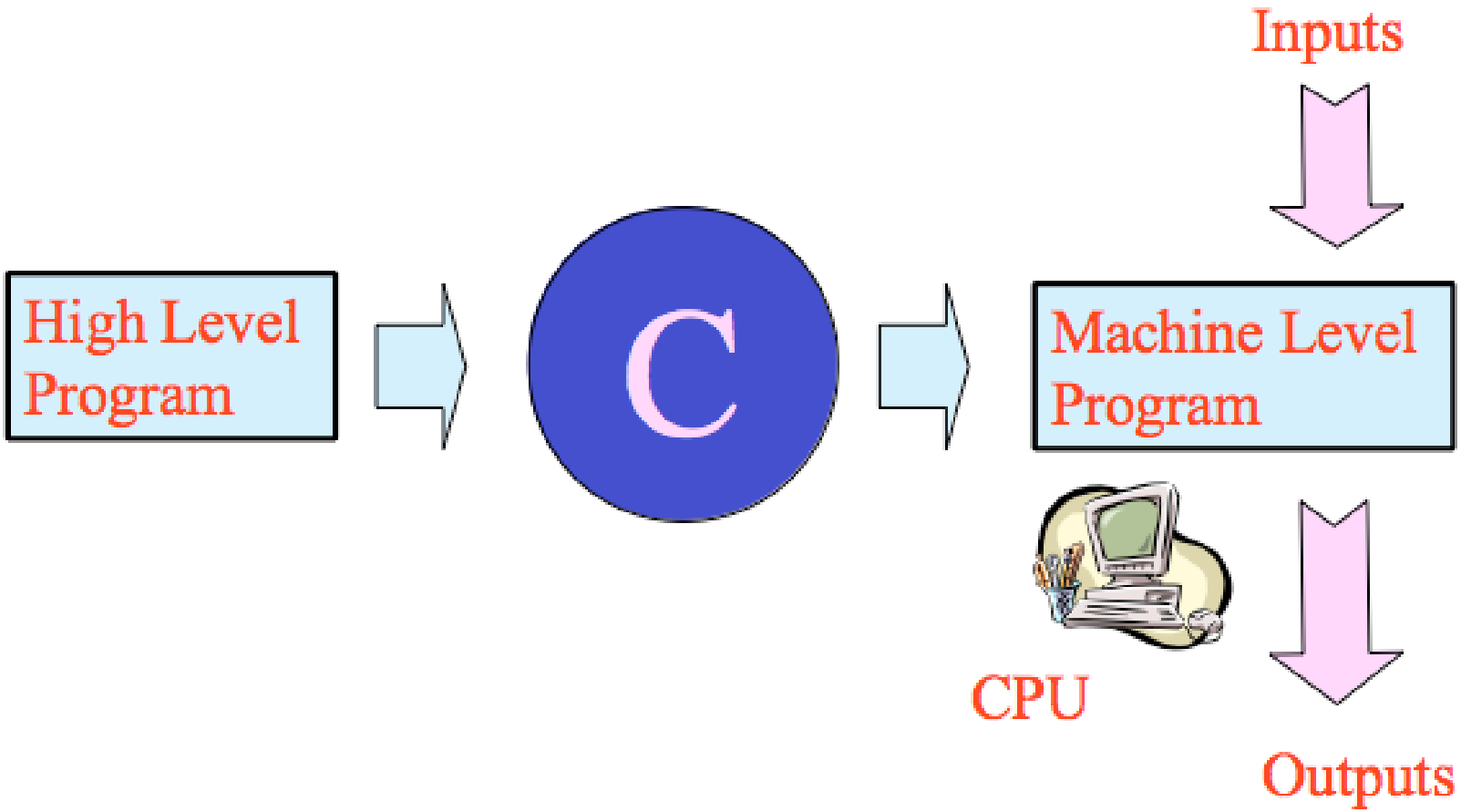
The interpreter is a machine level program, which interprets and executes the high level program, line by line.



(figure taken from the Scheme course site)

The Compiler

The Compiler translates the complete high level program to a machine level program.



(figure taken from the Scheme course site)

Compiled vs. Interpreted Programming Language

- The difference between a compiler and an interpreter usually reflects language difference.
- A compiler is useful if the language allows checking certain properties of the program before running it.
- An important main difference in this respect is between languages with static types and those with dynamic types.
- Python has dynamic types. The meaning of this will be discussed today or on Wednesday.
- It is generally believed that dynamic types give the programmer more flexibility, while static types provide more safety, because certain errors may be detected before running the program.
- In addition, interpreted languages are considered easier to debug.

Installing and Running Python 3.6

- Installing a Python 3.6 interpreter on CS machines in Schreiber: This was already done for you, courtesy of the CS system team.
- Installing a Python 3.6 interpreter on your own machine: Follow the website link http://tau-cs1001-py.wikidot.com/python. Alternatively, look it up on www.python.org, under Download.
- The TAs will not go over this in the recitation.
- Make sure the version you download fits your OS (Linux, Windows, MAC OS X) and word size version (32 bits or 64 bits). Windows 7 and 8 usually support word size of 64 bits.
- Note that we want a Python 3.6 interpreter. Previous Python 3.x versions (*e.g.* 3.2 or 3.4) are mostly OK, yet may cause problems with some packages we may use later. Python 2.x is completely incompatible with Python 3.x.
- If you use a Python 2.x interpreter (so writing Python 2.x programs), they will crash in our execution tests. This will have negative effects on the "wet" part of your homework assignments' grades, so is best avoided.

IDLE

There are many interfaces, or programming environments, for running Python. They supply different levels of support for catching bugs in Python code and for following execution dynamics, or debugging.

We will use one of the simplest such programming environment, called `IDLE`.

For large industrial projects, `IDLE` may be too simple. But it is completely adequate for the rather simple programs we (and you) will write in this course.

`IDLE` is good enough for the course staff, and we recommend you use it as well.

Python Programming Basics: "G'day, Mate"

The first line of code taught in all programming languages is a print command of a greeting.

We do not dare to deviate from this inspirational tradition, but will add an Aussie touch to it.

```
>>> print("G'day, mate")
G'day, mate
```

The text to the right of the prompt, >>>, is the "command" to the Python interpreter. The text in the next line is the value returned by the interpreter. This should actually be in blue, but the sophisticated system for type setting I'm using. LaTeX, fails to realize this.

`print` is a built-in Python "function or method". Python has a collection of reserved words, with fixed meaning. These are usually displayed using colors.

Read, Eval, Print

An interaction with the interpreter has 3 steps

- Read: the interpreter reads the sequence of characters we type following the prompt (and converts text to internal form).
- Eval: the interpreter evaluates (computes) the code that was read, and produces a result (and possibly additional effect).
- Print: the interpreter prints the result as a sequence of characters (converts internal form to text), then prints the prompt for the next interaction.

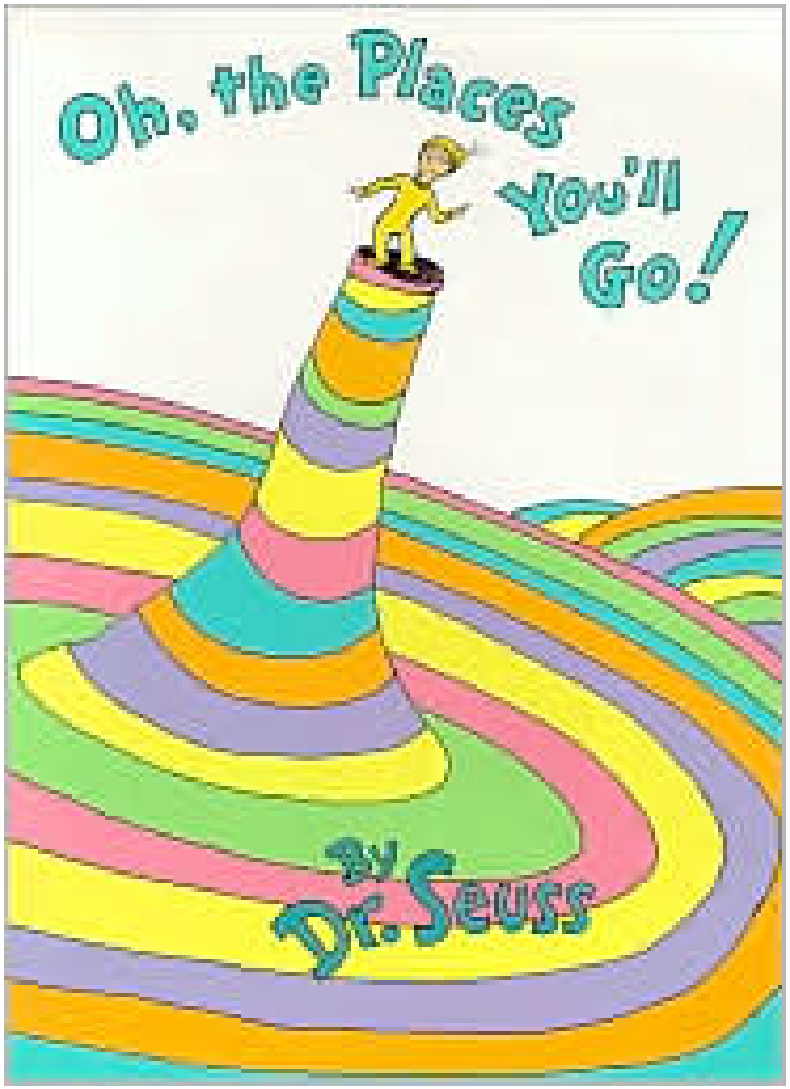
You Will Get Stuck!

I'm sorry to say so
but, sadly, it's true
that Bang-ups
and Hang-ups
can happen to you.

You can get all hung up
in a prickle-ly perch.
And your gang will fly on.
You'll be left in a Lurch.

You'll come down from the Lurch
with an unpleasant bump.
And the chances are, then,
that you'll be in a Slump.

And when you're in a Slump,
you're not in for much fun.
Un-slumping yourself
is not easily done.



What to Do When You Get Stuck?

1) Python interpreter has built-in `help` for all built-in and library functions/methods/classes. For example,

```
>>> help(print)
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file: a file-like object (stream); defaults to the current sys.
    sep:  string inserted between values, default a space.
    end:  string appended after the last value, default a newline.
```

Admittedly, `help` response may be somewhat cryptic at times.

2) Check Python documentation at http://docs.python.org/py3k/.

3) Use your favorite search engine. With high probability, any problem you ran into was already tackled by someone who documented the solution on the web.

4) The course forum may come in handy.

Python Programming Basics: Strings and Type 'str'

Let us explore this greeting a bit further.

```
>>> print("G'day, mate!")
G'day, mate!
```

Now let us see what happens if we omit the `print` command.

```
>>>"G'day, mate!"
'G'day, mate!' # the interpreter ''response'' is to
               # print the value of the last command
```

We now ask for the `type` of `"G'day, mate!"`

```
>>> type("G'day, mate!")
<class 'str'>
```

The answer we get is that it is of type str, indicatying this is a string. In python, a sequence of characters, enclosed by single or double quotes, is a string. Strings are colored green by the interpreter.

Strings have their own built-in methods, like converting to lower (or upper) case, replacing a substring by another, concatenation, etc. etc.

Examples of String Methods

Strings have their own built-in methods, like converting to lower (or upper) case, replacing a substring by another, concatenation, etc. etc. Some of these methods' names have `str.` as their prefix, indicating they operate on the class "string".

```
>>> str.upper("Benny")
'BENNY'
>>> str.lower("Rani")
'rani'
>>> str.replace("Real Men Don't Apologize",
        "Apologize","Eat Quiche")
"Real Men Don't Eat Quiche"
>>> "Py"+"thon"    # + denotes concatenation
'Python'
>>> "na "+"nach "+"nachman "+"nachman meUman"
'na nach nachman nachman meUman'
```

More String Methods

Strings have their own built-in methods, like converting to lower (or upper) case, replacing a substring by another, concatenation, etc. etc. Some of these methods' names have `str.` as their prefix, indicating they operate on the class "string".

```
>>> str.title("dr "+"suess")
'Dr Suess'
>>> str.title(str.replace("Life is Hell","Life","Garda"))
        #       composition of two string methods
'Garda Is Hell'
>>> "Bakbuk Bli Pkak "*4  # * denotes repetition
'Bakbuk Bli Pkak Bakbuk Bli Pkak Bakbuk Bli Pkak Bakbuk Bli Pkak '
>>> "Garda is hell"*0
''  # the empty string
>>> "Benny"+"Garda is hell"*0
'Benny'  # similar to adding 0 to a number
>>> "Garda is hell"*-3
''   # the empty string, again
```

There are obviously many other strings methods, but for the time being, these will do.

Python Programming Basics: Variables and Assignments

The following line is an assignment in Python. The left hand side is a variable. The right hand side is an expression.

```
>>> n=10
```

The interpreter evaluates the expression and assigns its value to the variable. Think of this assignment as creating a new "dictionary entry", where the variable's name, `n`, becomes bound to the value `10`.

The variable's name is a sequence of letters and digits, which should start with a letter. Underscore _ may also be used (to be discussed later in the course). Names are case sensitive: `My_Var` and `my_var` are not the same.

The Importance of a Name

What's in a name? that which we call a rose
By any other name would smell as sweet;

`William Shakespeare/Romeo and Juliet; ACT II, SCENE II`
(taken from John Guttag's, MIT 6.00SC intro CS course).

But in programming, names are important:
Programs should be readable by other programmers.

Variables and Assignments: An Example

```
>>> n=10
>>> n
10
```

The value can be changed by a subsequent assignment.

```
>>> n=11
>>> n
11
>>> type(n)
<class 'int'>
```

The type of the variable can change by a subsequent assignment. This feature is called dynamic typing (as opposed to static typing in other programming languages).

```
>>> n=1.3141
>>> n
1.3141
>>> type(n)
<class 'float'>
```

More Variables and Assignments

```
>>> e=2.718281828459045
>>> e
2.718281828459045
>>> pi=3.141592653589793
>>> pi
3.141592653589793
```

Variables with assigned values can be used as part of the evaluation of other expressions.

```
>>> e*pi+1
9.539734222673566
>>> e**pi+1
24.140692632779263  # ** stands for exponentiation
```

But assigning an expression with undefined variables leads to a run time error.

```
>>> e**(pi*i)+1
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    e**(pi*i)+1
NameError: name 'i' is not defined
```

More Variables and Assignments

But assigning an expression with undefined variables leads to a run time error.

```
>>> e**(pi*i)+1
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    e**(pi*i)+1
NameError: name 'i' is not defined
```

Remark: Leonhard Euler was familiar with the beautiful identity $e^{\pi \cdot i} + 1 = 0$ (aka the relation between the five most important numbers in mathematics).
But Python could not care less (unless we tell it what `i` is, namely assign a value to this variable).

Lecture 1: Highlights

- Variables belong to classes that determine their types.
- We saw the classes `'int'`, `'float'`, `'str'` (will see additional ones later).
- Different classes enable different operations.
- Some operations allow "mixing" variables of different types.
- Assignments require that the expression on the right hand side be already defined.
- Subsequent assignments to the same variable can change its value and even its type – this is called dynamic typing.
- Numbers of class `'float'` are real numbers, often approximating the full (infinite precision) value.