Extended Introduction to Computer Science
CS1001.py

Lecture 1: Introduction and Administratrivia
First Acquaintance with Python

Instructor: Benny Chor

Teaching Assistant (and Python Guru): Rani Hod

School of Computer Science
Tel-Aviv University
Fall Semester, 2011-12
http://tau-cs1001-py.wikidot.com
(it will take a few days for this site to actually work)

Travel Advisory

- You are about to take a new version of the "extended introduction to CS" course.
- This is the second run ever of this course.
- About half of first year CS students this year take this version.
- The older version of the course is still up and running.
- The programming language used in the older version is Scheme.
- The programming language used in the new version is Python.
- It is your choice (using the bidding system) which version to take (both have advantages and disadvantages).
- There are many good reasons to take the older version of the course.

Travel Advisory (cont.)

- There are many good reasons to take the older version of the course.
- The older version is much more stable, better documented and better supported than the new version.
- A large number of exams and homework assignments for the older version are available online.
- The older version has a universally acclaimed text book.
- The new version has none (acclaimed or not).
- The lecturer of the new version is erratic and unpredictable.
- But the teaching assistant more than makes up for this :=)

- Welcome aboard!

Administrative Details

- Intended for 1st year Computer Science students.
- Grade determined by exam (70-80%) and homework (30-20%).
- In order to pass the course, you must pass the exam and get a passing grade in at least $n-2$ out of the n home assignments.
- Homework grade is best $n-2$ out of n assignments.
- Exam on March 2nd, 2012 (Moed B on Marth 30th).
- Exam is closed book except for 2 double sided, normal size (A4) pages.

Administrative Details (2)

- 7–9 assignments, each with both "dry" and "wet" component.
- "Wet" homework submission in groups of size one or two (but not three or more).
- "Dry" parts of homework submitted in groups of size exactly one.
- Submissions after the deadlines will not be considered unless coordinated in advance (*e.g.* in case of reserve service) or reported asap and accompanied by valid documention (*e.g.* in case of serious illness).
- If one member of a pair has a valid reason for late submission, the other member is still expected to meet the deadline on his/her own.
- Office hours (both Benny & Rani): By e-appointment.
- E-mails: benny AT cs.tau.ac.il , ranihod AT tau.ac.il
- Course site: http://tau-cs1001-py.wikidot.com (not operational yet)

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 both you and your partner make a serious effort to solve the problems on your own.
- You may collaborate with people from other groups on the problem sets, but your solutions must be written up independently (by you for dry assignments, you and your partner only for code).
- 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 two sided 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" 10–14 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 both form integral parts of the course.

Programming 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.

- Those of you who heard about Scheme and were hoping to learn it here, should take the other intro CS class.

One More Thing this Course is Not About

Course Topics

(tentative list, not in order, somewhat ambitious)

- Python programming basics (2–3 meetings)
- Bits, bytes, and representation of numbers in the computer.
- Huge integers, with applications to public key cryptography.
- Representing and manipulating images.
- Text compression.
- Simple error correction codes.
- String matching.
- Hashing and hash functions.
- Comparing and aligning sequences, with applications in computational biology and in computer music.
- Numerical computations (Newton–Raphson root finding).
- Spatial pattern identifications via geometric hashing.
- Hard computational problems, and approaches to solving them.
- Secret sharing.

Course Site Lecture Notes

Lecture notes in pdf format (Adobe Acrobat) and Python code used in the class will be e-mailed to the course mailing list prior to the lecture.

Lecture notes after debugging will be posted on the course site. Likewise, Python code and logs of running them 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.

Guest Lectures

In addition to the regular classes, two (and possibly three) guest lectures are planned.
These are aimed at providing different viewpoints on topics related (directly or indirectly) to the course in particular, and to Computer Science in general.

Bibliography

There are many intro to CS textbooks out there.
But this newly devised course does not use any of them exclusively.
You can 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.
   Five copies of this book have already found their way to the library (519.836 ZEL).
3. Python 3.x 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 this summer (2011).

The 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 math. project).

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

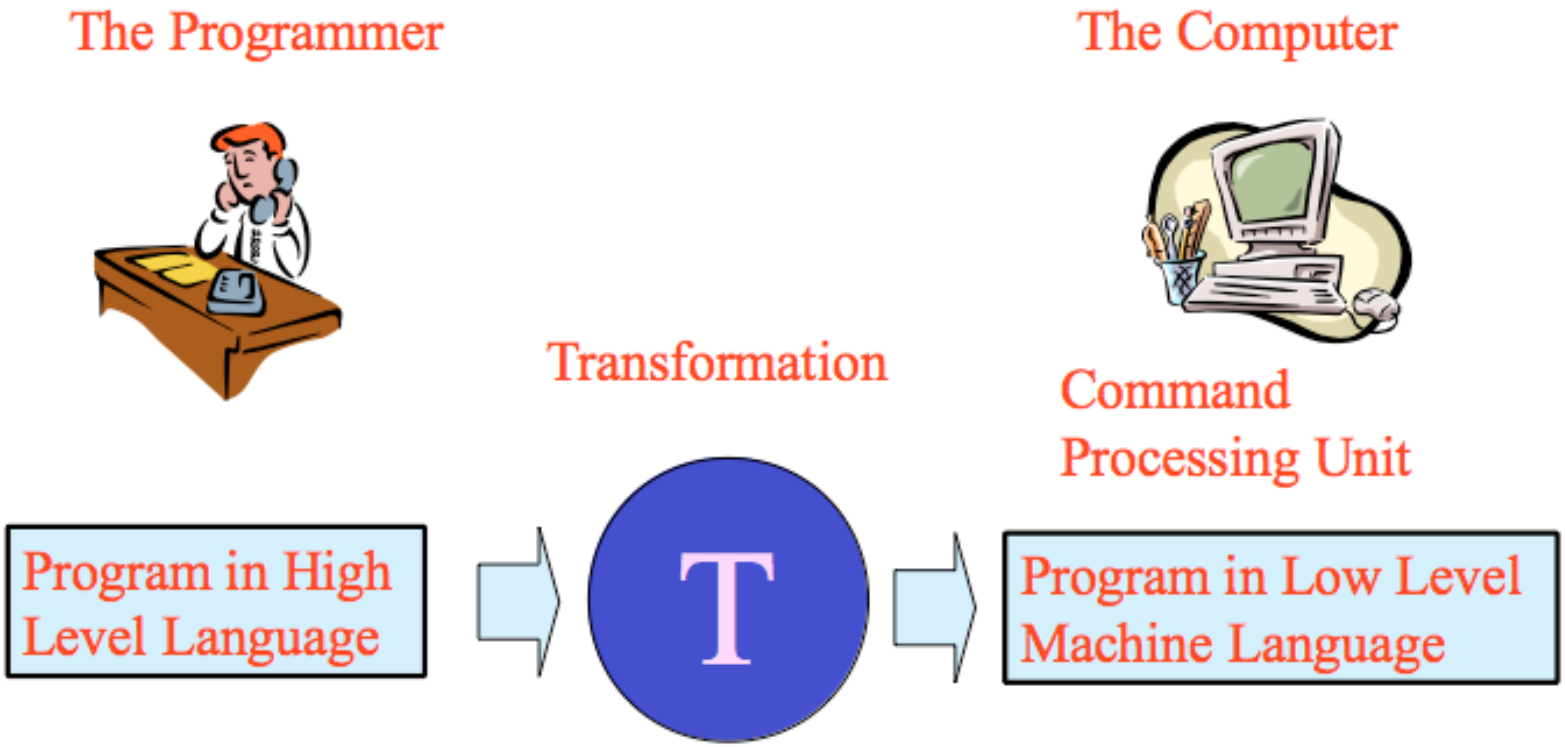
Programming Languages Basics

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.

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)

The Two Flavors of this Transformation

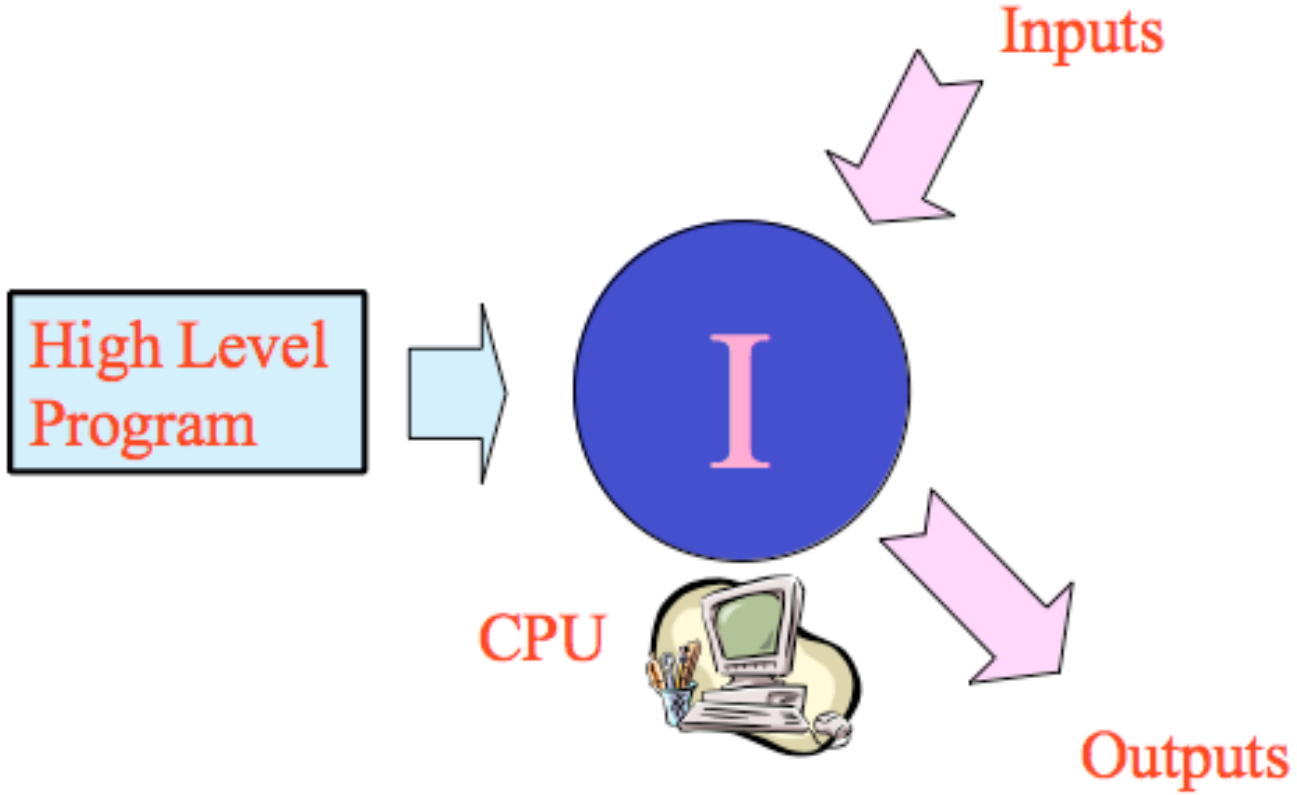
The transformation from high level to machine level languages comes in two flavors: By interpreters, and by compilers.

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.

The Interpreter

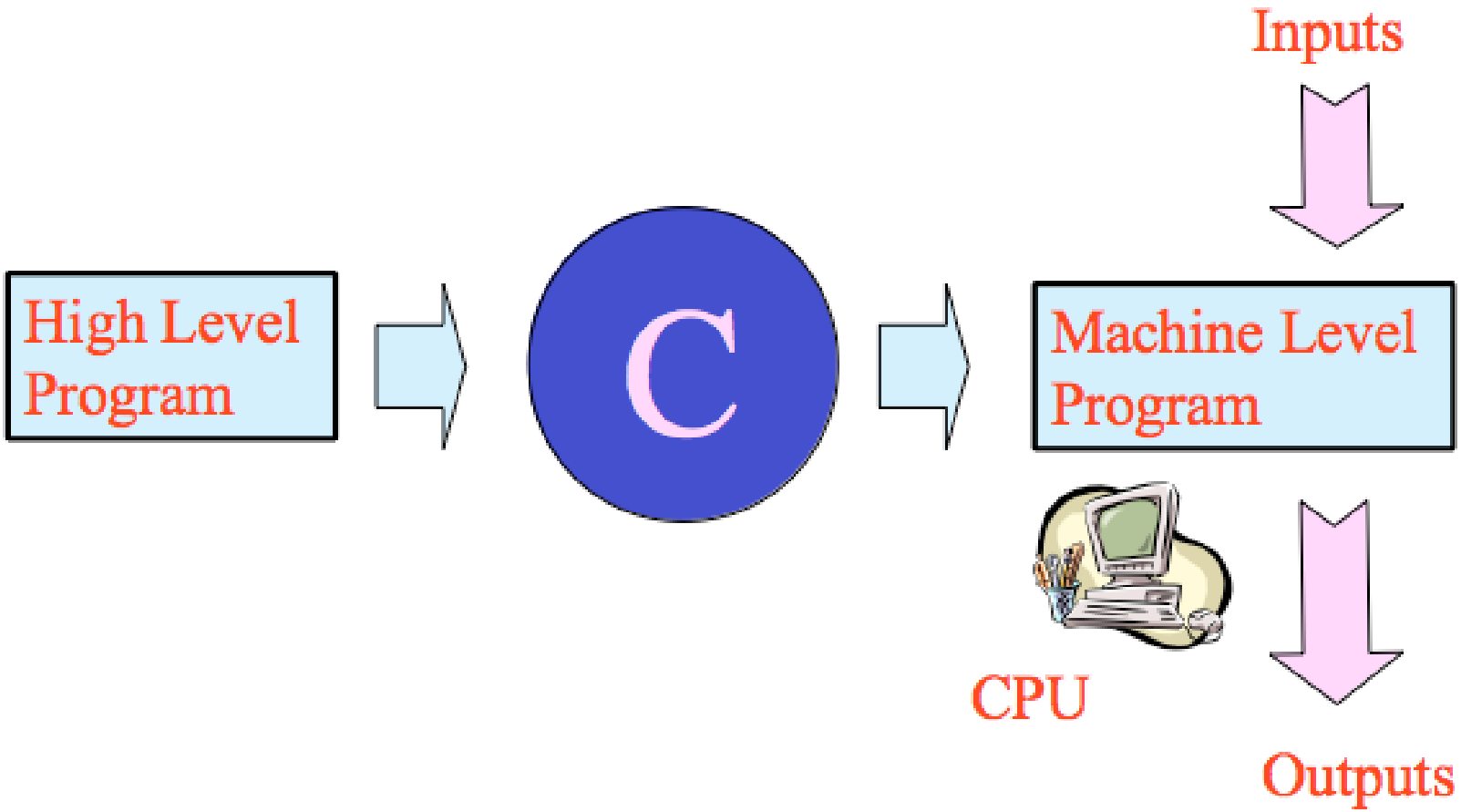
The interpreter is a machine level program, which interprets and executes the high level program, line by line.



(figure taken from the Scheme course)

The Compiler

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



(figure taken from the Scheme course)

Compiled vs. Interpreted Programming Language

Whiteboard discussion.

Installing and Running Python 3.x

- Installing a Python 3.x interpreter on CS machines in Schreiber: This was already done for you, courtesy of the CS system team.
- Installing a Python 3.x interpreter on your own machine: Simply look up Python's organization site the web, `www.python.org`. Download the interpreter from it, and install it using a few double clicks and maybe dragging an icon to some folder, depending on your OS (Linux, Windows, MAC OS X) and word size version (32 bits or 64 bits). The TA will go over this in the recitation.
- Note that we want a Python 3.x interpreter. Python 3.x is not fully compatible with Python 2.x.
- If you use a Python 2.x interpreter (so writing Python 2.x programs), they will most likely 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: "Gidday, 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("Gidday, mate")
Gidday, 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.

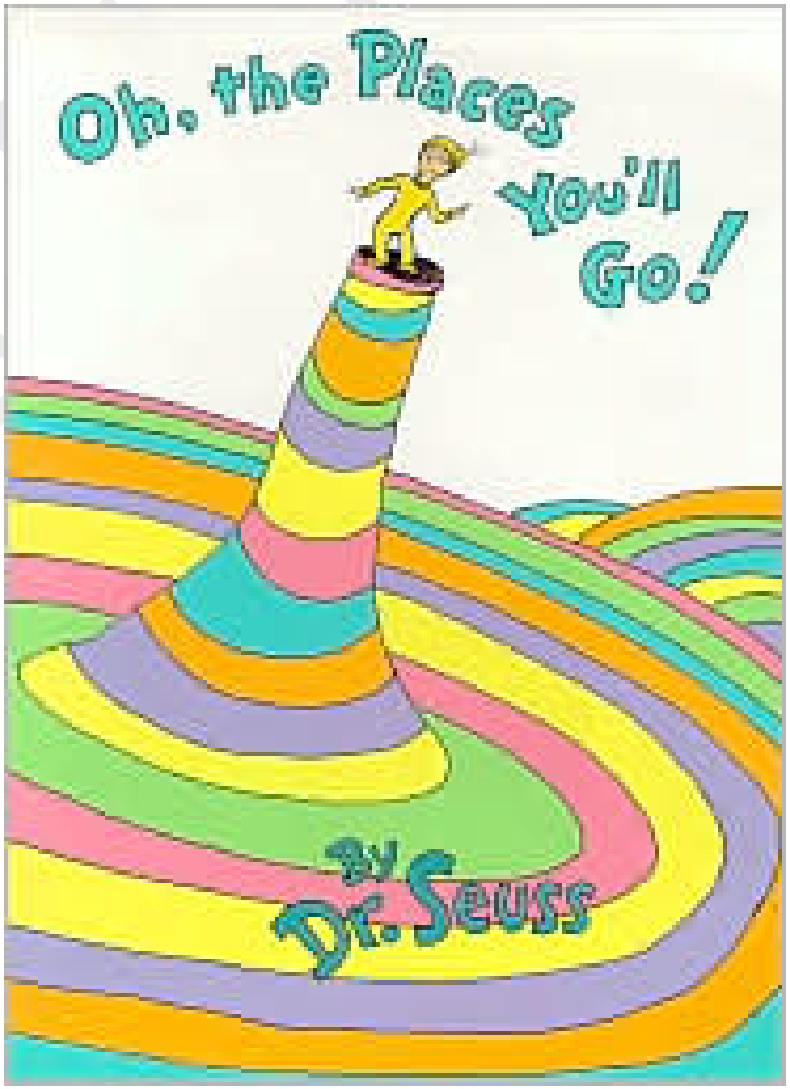
You Will Get Stuck!

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("Gidday, mate!")
Gidday, mate!
```

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

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

We now ask for the `type` of `"Gidday, mate!"`

```
>>> type("Gidday, 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"))
'Garda Is Hell'
>>> str.title(str.replace("Life is Hell","Life","Garda"))
        #       composition
'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: Numbers and their Types

```
>>> 4
4
>>> type(4)
<class 'int'>
>>> 3.14159265358979323846264338327950288
3.141592653589793  # ouch! truncated...
>>> type(3.14159265358979323846264338327950288)
<class 'float'>  # floating point ("reals") type
>>> complex(1,1)
(1+1j)
>>> type(complex(1,1))
<class 'complex'>  # complex numbers type
>>> 1j**2
(-1+0j)  # ** is exponentiation
>>> 8/5
1.6  #  / returns a float (real), the result of division
>>> 8//5
1  # // returns an integer, the floor of division
>>> type(8/5)
<class 'float'>
>>> type(8//5)
<class 'int'>
```

Addition, subtraction, multiplication exist as well (mix, match, try!).

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 key is the variable name, `n`, becomes bound to the value 10.

The variable's name is a sequence of letters and digits, starting with a letter.

Variables and Assignments: An Example

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

The value can be changed by a subsequent assignment.

```
>>> n=11
>>> n
11
```

The type of the variable can change by a subsequent assignment.

```
>>> n=1.3141
>>> n
1.3141
```

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*(0+1j))+1
1.2246467991473532e-16j # e**(pi*i)+1=0, up to small numeric error
# Last expression was somewhat mysterious. Let us demystify it:
>>> 10**(-16)
1e-16   # so previous outcome is approx. 1.22 * 10**(-16)
```

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

```
>>> e**(pooh*(0+1j))+1
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    e**(pooh*(0+1j))+1
NameError: name 'pooh' is not defined
```

Lecture 1: Highlights

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