Extended Introduction to Computer Science CS1001.py

Lecture 1: Introduction and Administratrivia First Acquaintance with Python

Instructors: Amir Rubinstein, Daniel Deutch

Teaching Assistants: Amir Gilad, Michal Kleinbort

School of Computer Science
Tel-Aviv University
Winter Semester, 2016-17
http://tau-cs1001-py.wikidot.com

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.
- 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.

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.

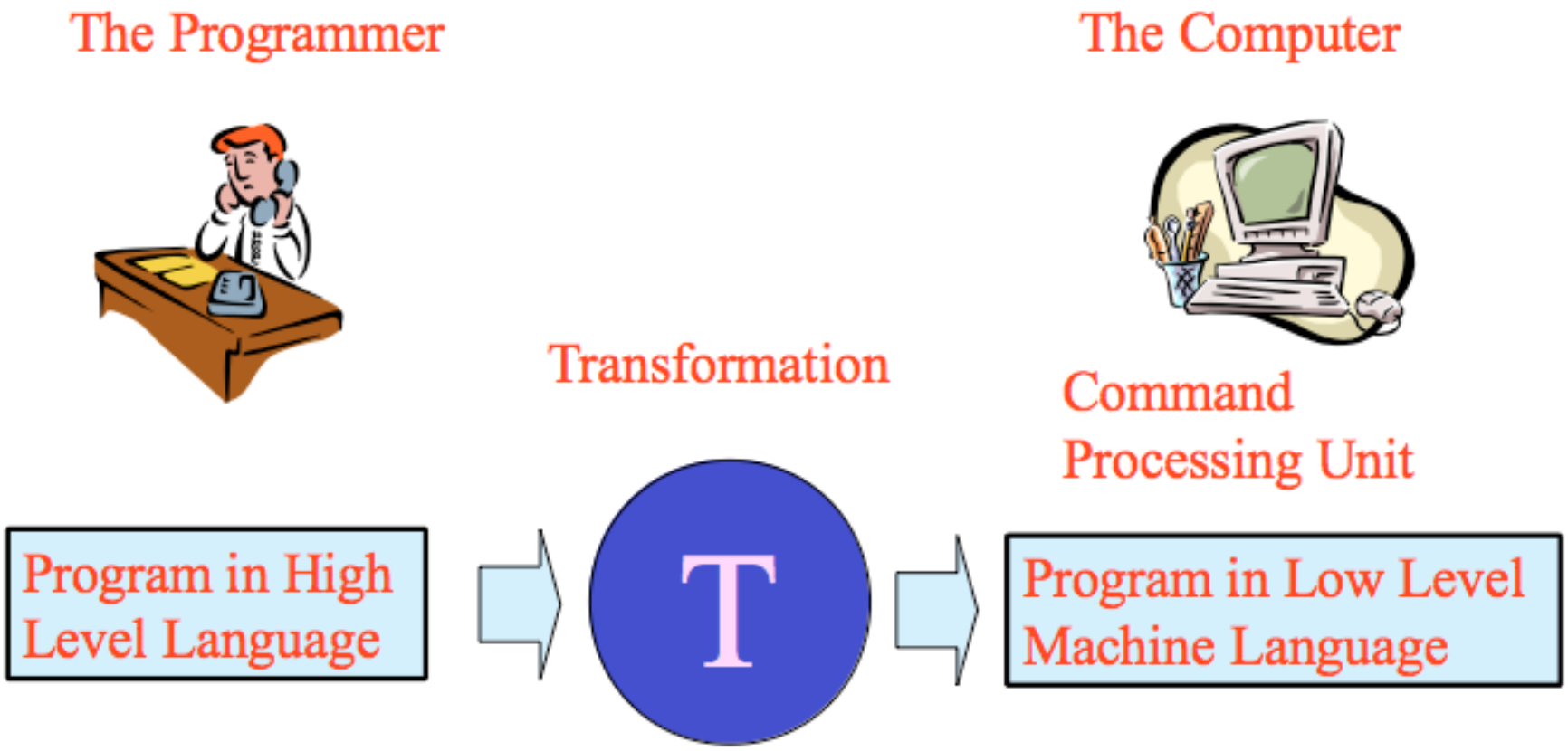
Writing Programs

- 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.

From High Level to Machine Level Languages



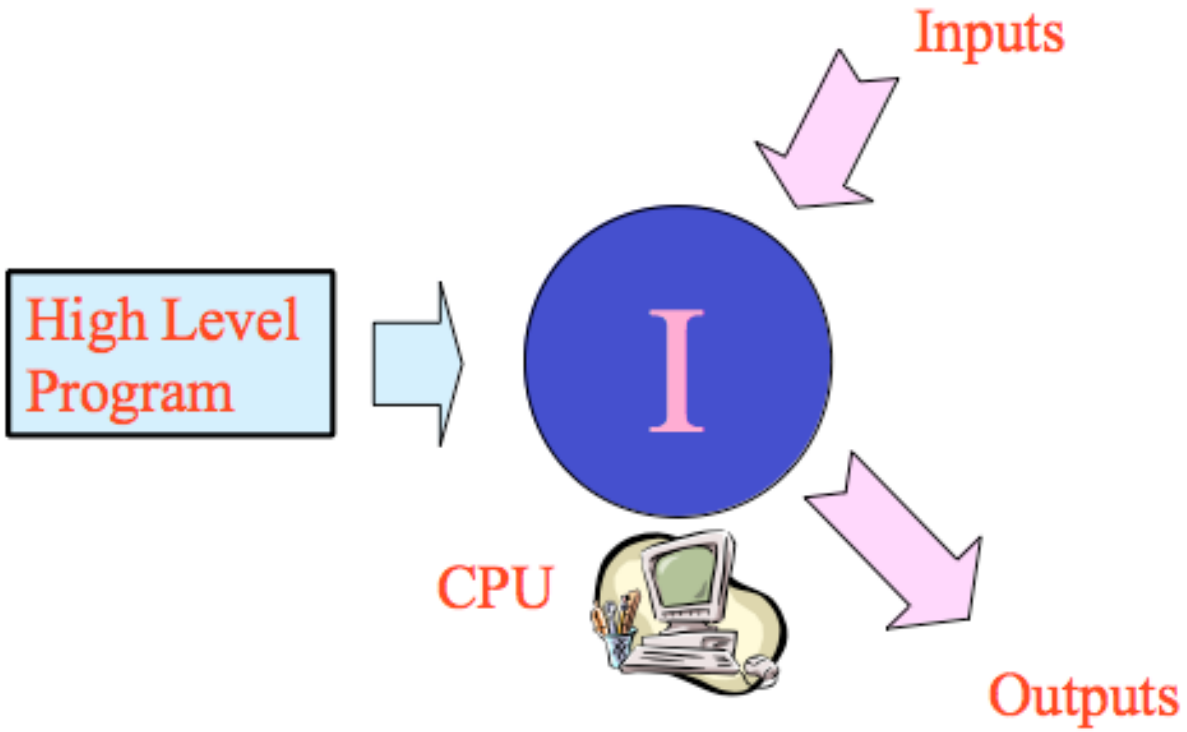
(figure taken from the Scheme course site)

The Two Flavors of this Transformation

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

The Interpreter

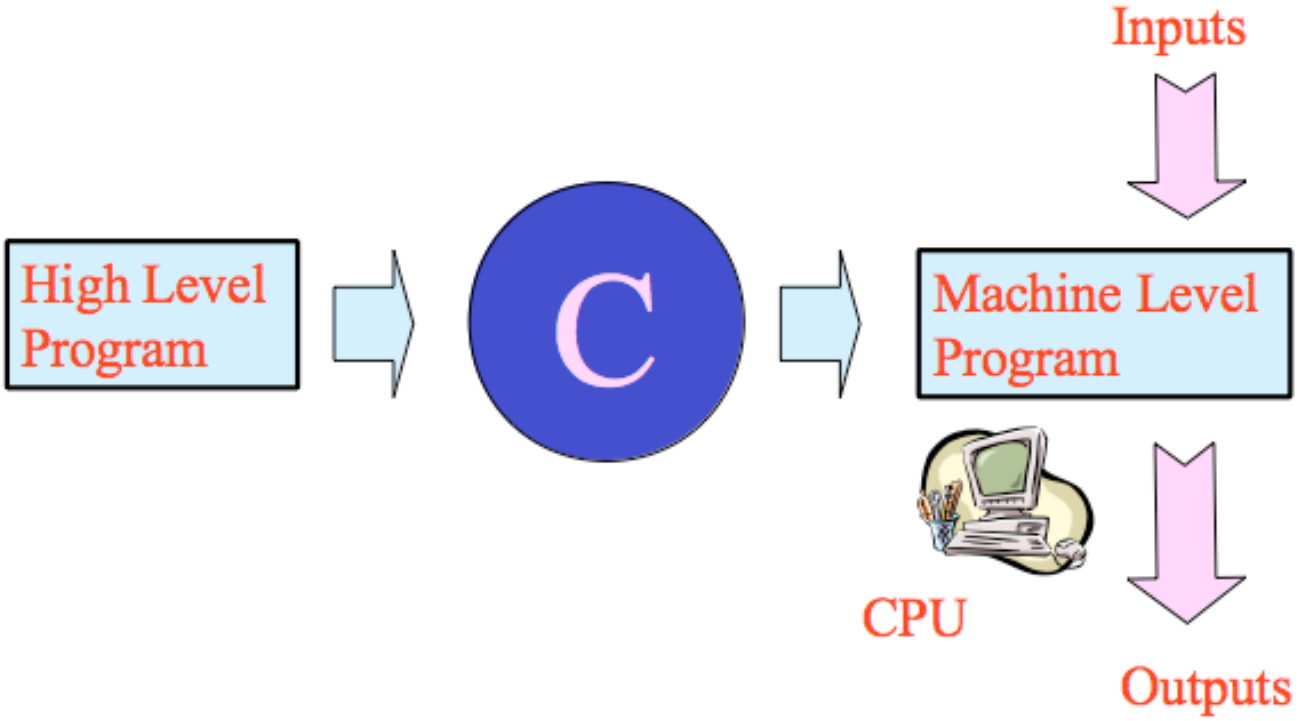
The interpreter is a machine level program, which interprets and executes the high level program, line by line.



(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)

Specific Programming Language

- Python is an interpreted programming language. And so are JavaScript, Lisp (and its variant, Scheme), MATLAB, Perl, PHP, Ruby, and many many other programming languages.
- In contrast, Java is a compiled programming language. And so are C, C++, Fortran, Haskell, Pascal, and many many other programming languages. [More precisely, Java is compiled to "bytecode", which is then interpreted]

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.
- The 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 understood later today.
- (It is 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.)

Installing and Running Python 3.4

- 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 3.4 interpreter on your own machine: Follow the website link http://tau-cs1001-py.wikidot.com/python . Alternatively, look it up on www.python.org
- 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 3.4.y interpreter. Python 3.x is not fully compatible with Python 2.x.
- If you use Python 2.x, your Python 2.x programs 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

print is a built-in Python function (colored purple by the interpreter). We will later see that Python has a collection of reserved words, with fixed meaning, usually displayed in red

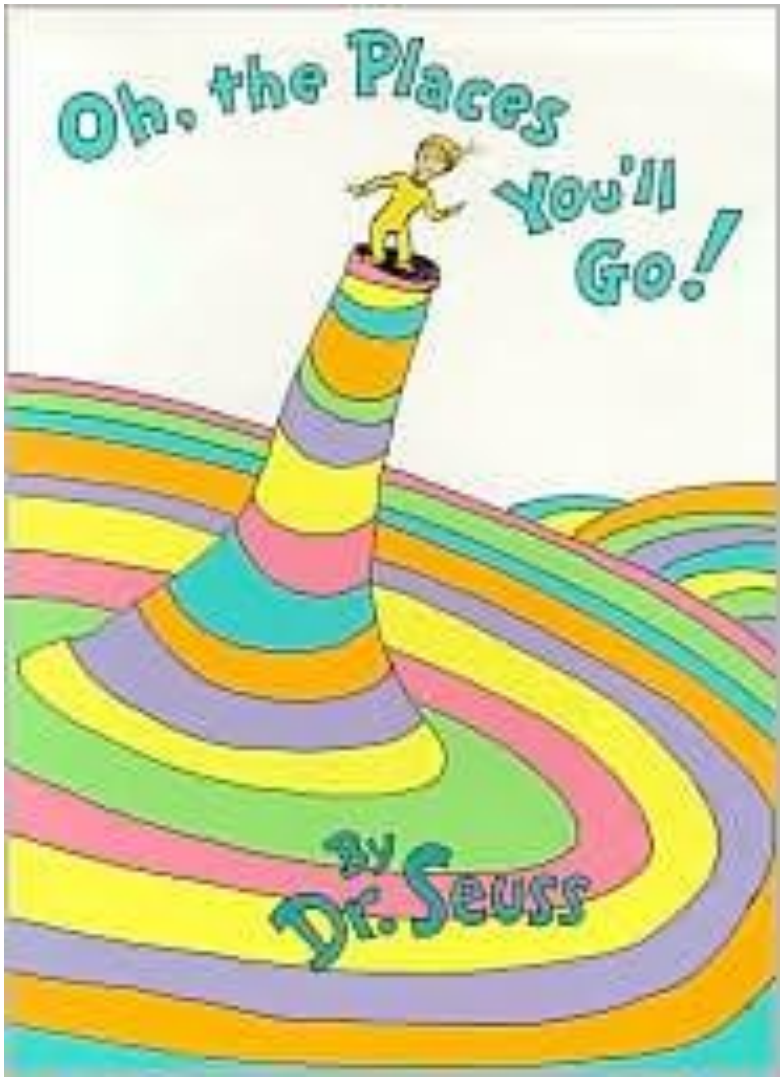
Read, eval, print

An interaction with the interpreter has 3 steps

- **Read**: the interpreter reads the sequence of characters we type following the prompt. (converts text to internal form)
- **Eval**: the interpreter evaluates (computes) the code that was read, and produces a result (and perhaps 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 (see next slide)

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.

Help example

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

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

    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.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

Python Programming Basics: Strings and Type ‘str’

>>> print ("Gidday , mate !")

Gidday , mate !

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

>>> " Gidday , mate !"

’Gidday , mate !’

The interpreter ‘‘response’’ -- prints the value of the last command

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

>>> type (" Gidday , mate !")

<class ’str ’>

It is of type str, indicating 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.

Examples of String Methods (1)

Strings have many built-in methods, like converting to lower (or upper) case, replacing a substring by another, concatenation, 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 '
```

Examples of String Methods (2)

```
>>> "Py"+"thon" # + denotes concatenation
'Python '
>>> "na "+"nach " + "nachman "+ "nachman meUman "
'na nach nachman nachman meUman '
```

The text that start with the hash character, #, and extend to the end of the physical line is a comment. (details later)

```
>>> str.title ("dr "+"suess")
'Dr Suess '
>>> str.title ( str.replace ("Life is Hell", "Life", "Garda"))
                    # composition of 2 methods
'Garda Is Hell '
```

Examples of String Methods (3)

>>> " 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.

Note function notation func(arguments) vs. infix operator notation

a + b

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.

In python, a variable is just a name. The variable's name is a sequence of letters and digits, which should start with a letter. Underscore _ may also be used (will be discussed later). Names are case sensitive: MyVar is different from myvar

The importance of names

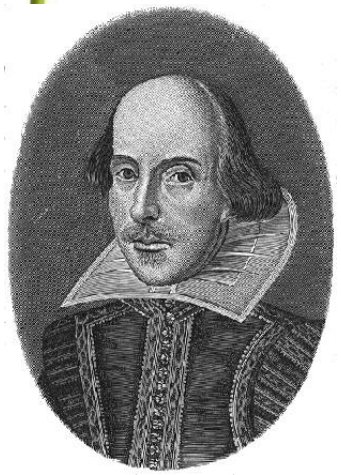
What's in a name? that which we call a rose

By any other name would smell as sweet;

מה יש בְּשם? מה שנקרא לו שושנה
ריחו יהיה מתוק תחת כל שם אחר

Shakespeare/Romeo and Juliet

ACT II, SCENE II



But In programming, names are important:

Programs should be readable by other programmers.

(taken from John Guttag's, MIT 6.00SC intro CS course).

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: so (as mentioned) in python the type of a variable is dynamic (as opposed to static types in other programming languages).

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

More Variables and Assignments (1)

```
>>> 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  # ** stands for exponentiation
24.140692632779263
```

More Variables and Assignments (2)

But 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
```

Glimpses Into the (Near) Future: Conditionals (if, elif, else)

```
>>>temp =30 # degrees centigrade
>>>wind =17 # knots ( nautical miles per hour )
>>>if temp > 25 and wind > 13:
        print ("go windsurfing ")
    elif temp > 25 and wind <= 13:
        print ("go to the beach ")
    elif temp > 30:
        print (" put your hat on")
    else :
        print (" attend class ")

Go windsurfing
>>>
```

Glimpses Into the (Near) Future: while

```
>>> n=100
>>> m=100
>>> while m>0:     # print all divisors of n
        if n % m == 0:
            print(m)
        m=m-1
100
50
25
20
10
5
4
2
1
```

Lecture 1: Highlights

- Values belong to classes that determine their types.
- We saw the classes 'int', 'float', 'str' (more classes later).
- Different classes enable different operations.
- Some operations allow "mixing" values of different types.
- An expression (eg. the right hand side of an assignment) must be defined – not contain undefined variables
- Subsequent assignments to the same variable can change its value and even its type.
  - We say that Python has dynamic types
- Numbers of class 'float' are real numbers, often approximating the full (infinite precision) value.